AD-A107 769   DELTA INFORMATION SYSTEMS INC JENKINTOWN PA   F/G 9/2
USE OF A MICROPROCESSOR TO IMPLEMENT AN ADCCP PROTOCOL WITH REJ--ETC(U)
AUG 81   DCA100-80-C-0221

UNCLASSIFIED   NCS-TIB-81-7   NL

1 OF 1
AD
A107769

END
DATE
FILMED
1-82
DTIC

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LEVEL II

AD A107769

19 REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

2

1. REPORT NUMBER 18 NCS-TIB-81-7	2. GOVT ACCESSION NO. AD A107769	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 Use of a Microprocessor to Implement an ADCCP Protocol with Reject & Selective Reject (Federal Standard 1003).		5. TYPE OF REPORT & PERIOD COVERED 9 FINAL REPORT,
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 15		8. CONTRACT OR GRANT NUMBER(s) DCA 100-80-C-0221
9. PERFORMING ORGANIZATION NAME AND ADDRESS Delta Information Systems, Inc. 310 Cottman street Jenkintown, PA 19046		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 11
11. CONTROLLING OFFICE NAME AND ADDRESS National Communications System Technology and Standards Office Washington, D.C. 20305		12. ~~REPORT DATE~~ AUG 1981
		13. NUMBER OF PAGES 83
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) 12 79		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution unlimited; approved for public release

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Telecommunications; Data Links; Microprocessors; ADCCP; Data Transmission Systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses the implementation of the reject and selective reject options of ADCCP (Advanced Data Communications Control Procedures) on an M6800 microprocessor. Flow charts and a portion of the code are included.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411214

TECHNICAL INFORMATION BULLETIN 81-7

USE OF A MICROPROCESSOR TO IMPLEMENT AN ADCCP PROTOCOL WITH REJECT & SELECTIVE REJECT (Federal Standard 1003)

AUGUST, 1981

APPROVED FOR PUBLICATION:

PREPARED BY:

Delta Information Systems, Inc
310 Cottman St
Jenkintown, PA 19046

Marshall L. Cain

MARSHALL L. CAIN
Assistant Manager
Office of NCS Technology
and Standards

FOREWORD

Among the responsibilities assigned to the Office of the Manager, National Communications System, is the management of the Federal Telecommunication Standards Program which is an element of the overall GSA Federal Standardization Program. Under this program, the NCS, with the assistance of the Federal Telecommunication Standards Committee, identifies, develops, and coordinates proposed Federal Standards which either contribute to the interoperability of functionally similar Federal telecommunication systems or to the achievement of a compatible and efficient interface between computer and telecommunication systems. In developing and coordinating these standards a considerable amount of effort is expended in initiating and pursuing joint standards development efforts with appropriate technical committees of the Electronic Industries Association, the American National Standards Institute, the International Organization for Standardization, and the International Telegraph and Telephone Consultative Committee of the International Telecommunication Union. This Technical Information Bulletin (TIB), one of a series, is a companion document to NCS TIB 80-7 (which incorporates all substantive material contained in NCS TIB 80-2), and has been prepared to inform interested Federal activities of the progress of these efforts. Any comments, inputs, or statements of requirements which could assist in the advancement of this work are welcome and should be addressed to:

Office of the Manager
National Communications System
ATTN: NCS-TS
Washington, D.C. 20305
(202) 692-2124

USE OF A MICROPROCESSOR TO
IMPLEMENT AN ADCCP PROTOCOL
WITH REJECT & SELECTIVE REJECT
(FEDERAL STD. 1003)

August , 1981

FINAL REPORT

Submitted to:

NATIONAL COMMUNICATIONS SYSTEMS
8th & S. Courthouse Road
Arlington, Virginia 22204

Contracting Agency:

DEFENSE COMMUNICATIONS AGENCY

Purchase Order: DCA 100-80-C-0221

Submitted by:

DELTA INFORMATION SYSTEMS, INC.
310 COTTMAN STREET
JENKINTOWN, PENNSYLVANIA 19046

TABLE OF CONTENTS

APPENDICES

1.0 INTRODUCTION

This document summarizes the work performed by Delta Information Systems, Inc. for the Office of Technology and Standards of the National Communications System, an organization of the U. S. Government, under Purchase Order DCA100-80-M-0221. The Office of Technology and Standards, headed by National Communications System Assistant Manager Marshall L. Cain, is responsible for the management of the Federal Telecommunications Standards Program, which develops telecommunication standards whose use is mandatory by all Federal agencies. The objective of this program is to develop a block diagram, flow charts, and computer programming for the Reject and Selective Reject functions of the unbalanced normal, unbalanced asynchronous, and balanced asynchronous class of procedures in accordance with Federal Standard 1003. The purpose of this effort is to determine the feasibility of using the M6800 or similar microprocessor to implement this type of protocol, and to obtain an estimate of memory and processor resources that would be required. The Office of Technology and Standards will use the information to advise other Federal agencies who implement the standard and, when merged with the results of other studies, to evaluate the operational and economic impact of incorporating various options in Federal Standard 1003.

The effort necessarily has focussed on the software required to implement the protocol itself , and is by no means a total hardware/software system design that would be required to develop a complete system. Complete system development is, of course, beyond the scope of this program.

A discussion of the method of implementation of Reject and Selective Reject is included in Section 2. Flow charts describing the software that makes up the protocol included in Section 4. These flow charts describe the protocol software processes in sufficient detail that code may be written with no major design decisions. The flow charts at this level are very hardware dependent.

A small portion of the code for the 6800 microprocessor has been written and is included in Section 5.0. The code was intorduced into a 6800 microcomputer, provided by Delta Information Systems. The code in the computer was then tested to insure its validity. Finally Section 6.0 contains a discussion of the feasibility of using the 6800 to implement the ADCCP protocol. It is estimated that approximately 1300 instructions are needed to implement the three classes of procedures in a logically configurable station (primary, secondary, or combined) with the Selective Reject option (1200 instructions with the Reject option), and that approximately 250 instructions are required for the operating system. Data transmission rates of up to 19.2 kilibit/sec. appear feasible for the configuration being considered.

2.0 SYSTEM DESIGN CONSIDERATIONS

The block diagram in Figure 2-1 shows a link with one primary/combined and one secondary/combined station communicating with each other by sending information in both directions. That is, either station may be a source or sink of data or both. Two-way simultaneous transmission is assumed. Although many secondary stations may communicate with one primary station, the objectives of this program can be met with no loss of generality, by assuming the existence of only one secondary station.

Each station, primary, secondary, or combined is made up of a microcomputer, an LSI interface to the link, and a user which supplies and uses the data to be communicated. The primary and secondary stations are physcially very similar; operationally, of course, the primary must supervise and control a number of secondary stations, and thus it requires a larger data structure and somewhat more complicated code.

For the purpose of this program, the microcomputer can be assumed to be very basic--microprocessor, memory (RAM and ROM), interface chips, clock, etc. A design choice that has significant impact on the outcome of this program is the choice of the LSI interface. The purpose of the LSI interface is to convert the parallel data from the CPU to a continuous serial data stream for transmission. Simultaneously, it must convert received serial data to parallel data for the CPU. In addition, it must generate and verify the frame check

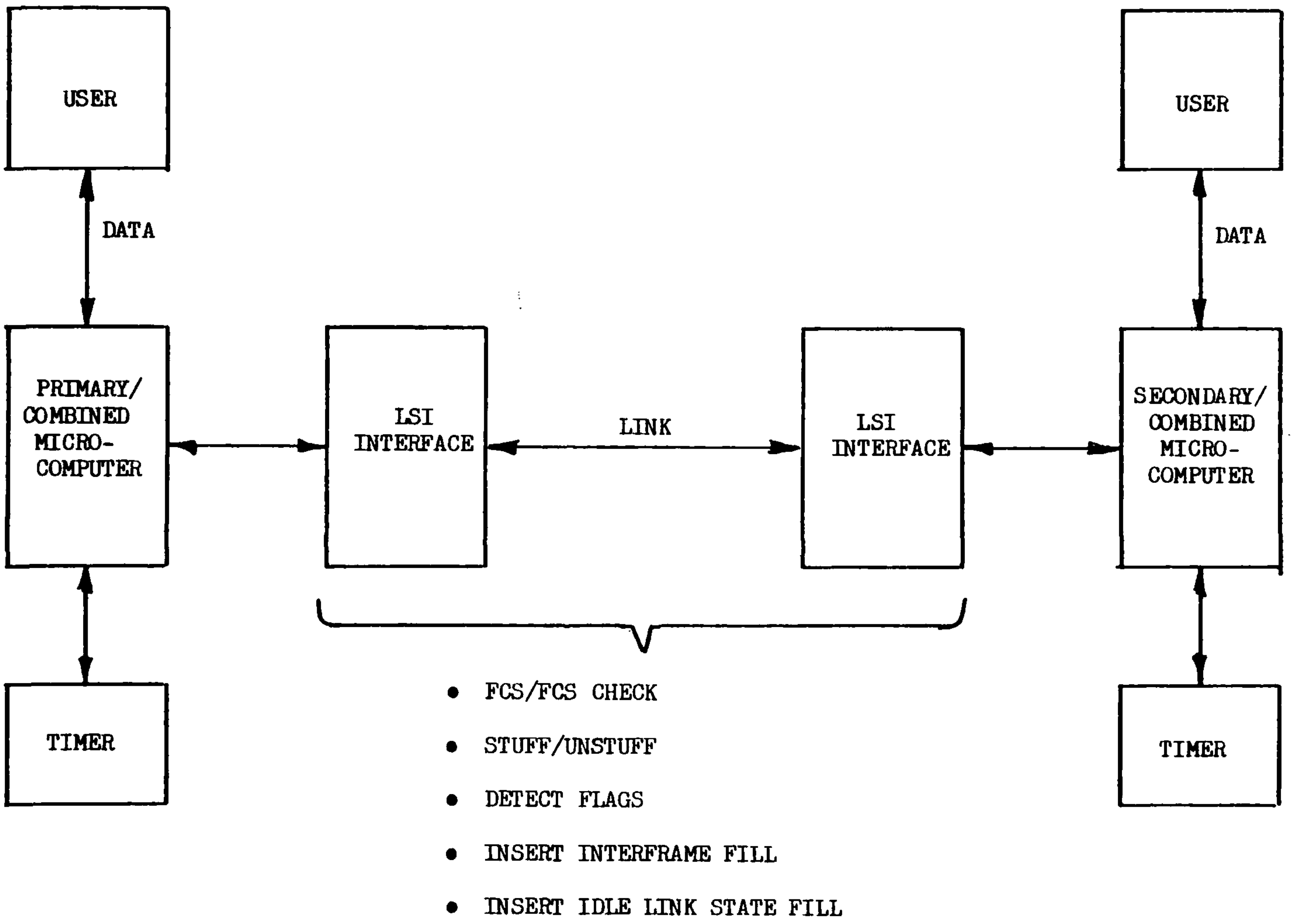


Figure 2-1 System Block Diagram

sequence (FCS), stuff and delete 0's to distinguish FLAG or ABORT from data, insert and detect FLAG or ABORT, and insert interframe fill or idle link fill. Other functions may also be performed by this interface.

Two different LSI chip specificiations have been examined as possible candidates for the interface function in this particular study. These chips, which represent different approaches to the interface problem, are the F6856 and the WD2501. (Refer to a previous report (1) for a copy of their preliminary data sheets and a brief comparison).

At least three other similar protocol chips have become candidates since the beginning of this effort. These are the Solid State Scientific SND5025, the Signetic 2652 and the Zilog SIO.

The F6856 chip was selected for this program by mutual consent of the contractor and the government. The interface to the communications lines requires additional logic such as a Federal Standard 1031 (Electronic Industries Association Recommended Standard 449) interface chip and a modem, but the choice of these has little impact on this program.

The data transmitted over the link, must also be transmitted to the user. The interface/protocol required between the microcomputer and the user is also part of the system design. However, for this program, the protocol has not been defined. The interface, including the buffers to hold the data, is defined and described in Section 4.0.

3.0 REJ/SREJ IMPLEMENTATION

The objective of this program is to add the Reject (REJ) and the Selective Reject (SREJ) functions to the design of the ADCCP according to the Federal Standard 1003 and to estimate the memory and processor resources that these functions require. The REJ and SREJ protocols may be used in a number of ways: pure REJ, pure SREJ, and various combinations[2] of these two including a look-ahead SREJ/REJ. For the pure REJ protocol, the receiver of information frames uses the REJ supervisory frame to request the sender of information frames to back up to an earlier point and begin retransmission. On the other hand, SREJ requests a single previous information frame followed by the continuation of the send sequence. REJ and SREJ may be used together in various combinations depending on whether the lost frame is single or multiple and if an exception condition exists when a new loss is detected.

The addition of REJ and SREJ to the ADCCP protocol also adds some complexity (especially for SREJ). This is due to the somewhat redundant nature of the reject protocols with respect to the checkpointing and timeout mechanisms. Although the reject mechanism is more efficient, checkpointing must be operational as a backup in case the REJ or SREJ frames or the retransmitted information frames are lost due to errors. Care must be taken to ensure that the checkpointing mechanism works with the reject frames while preventing redundant retransmission.

Only the pure REJ and pure SREJ have been included in the design, because these are adequate to allow memory requirements and speed to be estimated. However, the software design has been reorganized to provide for pure REJ, pure SREJ, or a combination of these options for error recovery. This has been accomplished by the inclusion of a recovery module in the subroutine that receives information frames. As described in more detail farther along in the report, this module is programmed for the desired recovery option. If a more complex recovery option is desired such as a look-ahead REJ/SREJ option, most of the software for this option will reside in the recovery module and the other modules will suffer little change.

4.0 DETAILED SOFTWARE DESIGN

In this section the software design is presented in sufficient detail to allow the objectives of this program to be met: that is, the feasibility of using the 6800 and an estimate of memory and throughput can be obtianed. The design covers the major aspects of a logically configurable station; functions that allow the station to operate as a primary, secondary, a combined station in unbalanced normal, unbalanced asynchronous or balanced modes are included. Operation as a secondary/combined station is emphasized, since FED-STD-1003 does not cover many of the primary/ combined station procedures for managing the link. These are left to the system designer. The software design includes a description of a minimal operating system to handle concurrent processes, major data structures, major software routines, and a set of detailed flow charts.

The detailed flow chart, together with associated data structures, describes the protocol software processes in sufficient detail so that code may be generated with no major design decisions. The flow chart at this level is hardware dependent, and must take into consideration the time constraints imposed by the concurrent software processes associated with the implementation of the protocol.

The protocol is made up of four major concurrent software processes, each of which is an example of the classic producer/

consumer problem. In this problem, one process produces items and then deposits them into a buffer. A second process consumes the items by taking them from the buffer. The processes must be coordinated so that the consumer does not run ahead of the producer, and that the producer does not write over records before the consumer has had a chance to read them. For the protocol problem, two concurrent processes are involved in communicating data between the LSI interface and the microprocessor; the LSI chip deposits bytes in its buffer as the producer, and the MPU reads this data as the consumer. Conversely, the microprocessor writes data into a buffer as the producer, to be read by the LSI chip as the consumer and transmitted over the link. A similar pair of processes serves to implement the interface between the microprocessor and the higher level user. For this effort, emphasis is placed on the interface between the MPU and the LSI protocol chip. This requires two main processes to be running at the same time--transmit and receive. The operating system that manages these processes is presented in Appendix A.

The purpose of this phase of the program is to add the REJ and SREJ functions to the design and estimate the memory and processor resources that these functions require. In order to accommodate REJ and SREJ in a modular fashion, the routines that receive I frames and that transmit I and supervisory frames have been reorganized and supplemented. For example, the receive I subroutine now includes a recovery module that can be used for REJECT, or SREJECT, or neither depending on which of three software routines are included. Note that either pure REJ or pure SREJ is included and not a combination of the two. The combination could be added as a fourth recovery technique, although for the purpose of measuring memory and processor resources the pure REJ and SREJ provide an adequate vehicle. If neither REJ or SREJ are employed the error recovery defaults to the checkpointing mechanism.

In addition to the receive I frame routine, the process that transmits I and supervisory frames was modified to enable REJ and SREJ transmission and retransmission when appropriate. The ability to reset the send variable as a result of receiving REJ was provided as well as the ability to transmit a selected I frame as a result of receiving SREJ. Two new routines were added to act on a received REJ or SREJ.

4.1 DATA STRUCTURES

This section outlines the data structures, including variables, arrays, buffers, etc. in order to more easily understand the detailed flow charts that follow and to estimate the amount of random access memory (RAM) that is required to implement the protocol. Main state variables are as follows:

STATION TYPE - PRIMARY, SECONDARY, OR COMBINED (mutually exclusive)

OPERATIONAL STATE - has 3 values, mutually exclusive:

LDS - Logically disconnected state

ITS - Information transfer state

FRMR - Frame reject state (for secondary/combined)

Logically disconnected state has two mutually exclusive modes:

NDM - normal disconnected mode

ADM - asynchronous disconnected mode

Information transfer state has three mutually exclusive modes:

NRM - normal respond mode

ARM - asynchronous respond mode

ABM - asynchronous balanced mode

Other major variables are:

REMOTE BUSY - true if RNR received
false if RR received or P/F bit
set in received I-frame

STATION BUSY - true if not prepared to receive
information; false otherwise

SNDREJ/ SNDSREJ - flag

REJ/SREJ ACTIONED - state variable

SENT REJ/SREJ - state variable

P-BIT - Poll bit

F-BIT - final bit

S - Send Variable (next I-frame to be transmitted)

R - Receive Variable (expected sequence number of next received I-frame)

N(S) - Send Sequence Number (I-frame sequence number)

N(R) - Receive Sequence Number (station transmitting N(R) has correctly received all I-frames up to and including N(R)-1)

Operating System variables include:

RDAFLG - Receive data available event variable

TBMT - Transmitter buffer empy event variable

RUNNING
BLOCK } Pointers to beginning of each queue
READY

PCB - Each process control block contains:

- condition code register
- accumulator A
- accumulator B
- index register (upper and lower)
- program counter (upper and lower)
- pointer to next PCB
- priority

A number of buffers are required for such things as the received control word, transmitted control word, frame type,etc. Next, consider the data buffer required to transmit/receive information between CPU and USER. Assume that a separate buffer is required for tansmit and receive, and that each buffer can hold up to eight I-frames of data. These buffers are accessed via tables shown in Figure 4-6. Each frame to be transmitted via the LSI chip has a starting address for the data and length in bytes of the data part of the frame. If the frame was transmitted with the poll/final bit set, this is recorded. The "acknoweldge" variable is used to indicate whether a frame has been deposited by the USER for transmission, whether it has been transmitted, and finally, whether it has been acknowledged by the receiving station. In the example shown, six frames have been deposited by the USER for transmission, three have been transmitted ending with a final bit (SECONDARY-NRM), and the first frame has been acknowledged. Three frames remain to be transmitted. The received look-up table performs a similar function for data received from the LSI chip. Each frame is assembled byte-by-byte and the frame length is incremented. When the frame has been correctly received (valid FCS and N(S)) the frame is tagged as verified and may be read by the user.

The buffers and associated variables required for LSI Interface chip operation are shown in Figures 4-2, 4-3, and 4-4. The Mode Control Register (MCR) contains control information common to both receiver and transmitter. The SAR contains the secondary station address. The TCR is loaded by the MPU to control the transmitter, and the TDB contains the byte to be transmitted. The Receiver Status Register (RSR) is read by the MPU to determine the status of the byte received in the Received Data Buffer (RDB). The RCR contains control information for the receiver and the TSR supplies transmitter status. Refer to previous report for a detailed description of receiver/transmitter operation and flow charts for the F6856 LSI interface chipe

4.2 SUBROUTINE/FUNCTION DESCRIPTIONS

This section describes the functions of software modules that make up the protocol. Figure 4-5 contains a table of all the significant software modules organized by station types (primary, secondary, or common to both) and by whether the module is associated primiarly with a transmit

TPACK

FRAME NUMBER	STARTING ADDRESS	FRAME LENGTH	FINAL BIT SET	ACKNOWLEDGEMENT
0	ADDR0	1028	0	1
1	ADDR1	1028	0	0
2	ADDR2	512	F	0
3	ADDR3	512		-1
4	ADDR4	512		-1
5	ADDR5	512		-1
6				8
7				8

TRANSMITTER LOOK-UP TABLE

RPACK

FRAME NUMBER	STARTING ADDRESS	FRAME LENGTH	FRAME VERIFIED
0			
1			
2			
3			
4			
5			
6			
7			

RECEIVER LOOK-UP TABLE

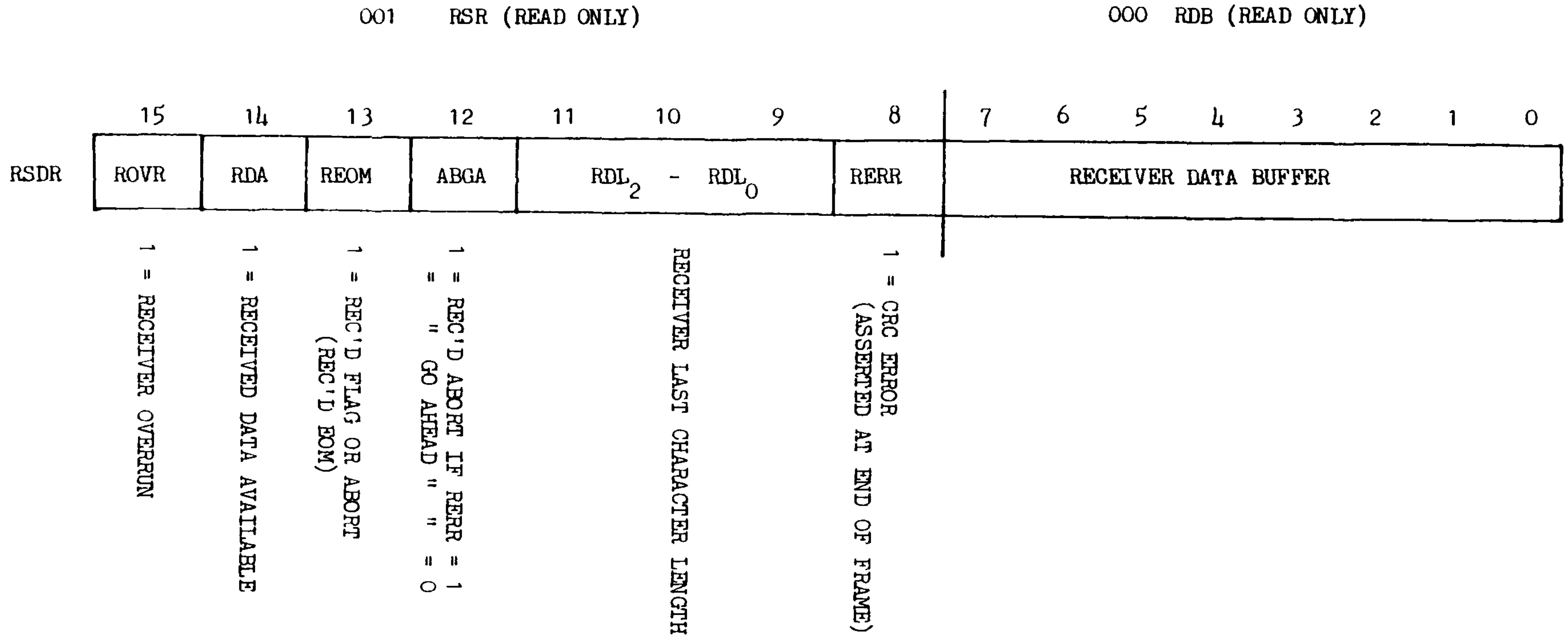
Figure 4-1 Transmitter/Receiver Look-up Table

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	101 MCR (WRITE ONLY)								100 SAR (WRITE ONLY)							
MSCA	PROTOCOL SELECT			LRSS	CC	NRZI	LOOP	EC	SECONDARY ADDRESS							
DEFAULT:	0 BOP	0 PRIMARY	0		0 1 CONT. BYTE	1 NRZI	--	1 CRC '1's								
SET:	0 BOP	1 SECONDARY	0	0 NORMAL MODE	0		--	1	STATION SECONDARY ADDRESS							

	15	14	13	12	11	10	9	8	7–0
	011 TCR (WRITE ONLY)								010 TDB (WRITE ONLY)
TCDR	SOM	TACG	GATD	EOM	RTS	TCL_2 - TCL_0			TRANSMITTER DATA BUFFER
(DEFAULT=0's)	1= START OF MESSAGE	1= ABORT	0= FLAGS TRANS. BETW. FRAMES	1= CONT. OF TDB IS LAST BYTE OF MESSAGE	1= REQUEST TO SEND	0 8-BIT TRANS. CHAR. LENGTH	0	0	

MUST BE RELOADED EACH TIME TCR IS
UPDATED UNTIL AFTER EOM BIT HAS BEEN SENT

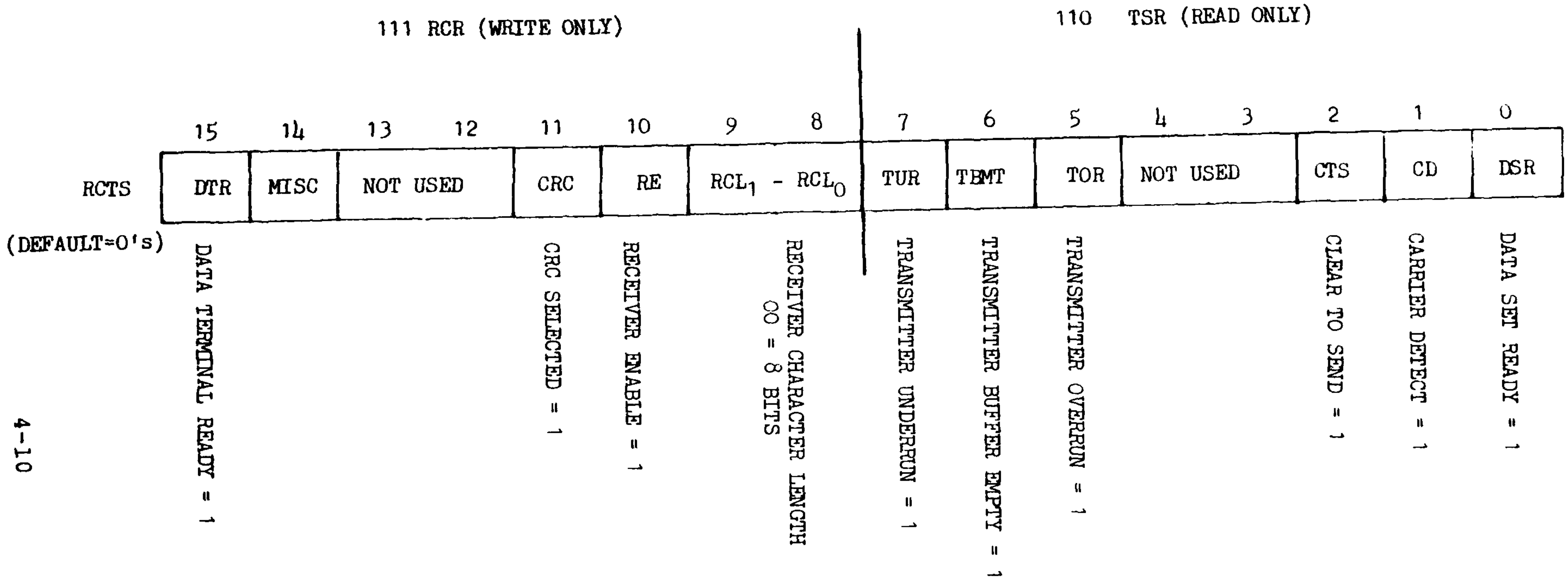
Figure 4-2 Mode Control, Secondary Address, Transmitter Control, and Tranmitter Data Registers



ALL BITS OF RSR EXCEPT RDA ARE RESET ON READ; RDA IS RESET WHEN RDB IS READ

DATA IS PASSED TO RDB ONLY ON ADDRESS MATCH

Figure 4-3 Receiver Status Register and Receive Data Buffer



TOR, TUR ARE RESET WHEN TSR IS READ

TBMT IS RESET WHEN TDB IS LOADED

Figure 4-4 Receiver Control Register, Transmitter Status Register

SECONDARY/COMBINED

RECEIVE FUNCTIONS	TRANSMIT FUNCTIONS
SNRM	
SARM	TR-UA
SABM	TR-DM
DISC	
RESET	
FRMREJ	TR-FRMR

PRIMARY/COMBINED

RECEIVE FUNCTIONS	TRANSMIT FUNCTIONS
	TR-SNRM
UA	TR-SARM
DM	TR-SABM
	TR-DISC
FRMR	TR-RSET

BOTH

RECEIVE FUNCTIONS	TRANSMIT FUNCTIONS	OPERATING SYSTEM
RCV	SENDI	WAIT
I	SNDRR	SIGNAL
RR	SNDRNR	INTERRUPT SERVICE
RNR	SENDRJ	IDLE
CHKPNT		DUMMY
RCNTRL		
GETBYT	SNDBYT	
REJ		
SREJ		

INIT

Figure 4-5 Software Modules

or receive process. Those modules contained in the operating system have been discussed previously. Some of the modules are the main routines for processes, namely INIT, RCV, SENDI, and IDLE. INIT is the intialization process, RCV, is the process that receives and processes the address and control bytes of the received frame, SENDI is the main transmitting process, and IDLE is a simple process that runs when all other processes are blocked. DUMMY is a process that is never run, but serves as a place for the IDLE process to point to and has the lowest priority. Those modules that are associated only with either the primary/combined or secondary/combined stations are mostly mode-setting or mode acknowledging functions and are relatively simple.

GETBYT is a macro that reads the receive buffer if data is available and stores the data in a location specified in the macro agreement. Interrupts are disabled for the duration of the macro to prevent flags or data from changing while reading. If no data is available, the process is blocked via WAIT. The SNDBYT macro loads the transmitter buffer with operation similar to that in GETBYT.

FRMREJ changes the state of the secondary/combined station to frame reject, assembles the FRMR information field, and activates the FRMR transmit process called TR-FRMR. This process terminates any other transmit process except TR-DM, or TR-UA, sends the FRMR frame, and then terminates itself. The FRMR routine receives a FRMR frame (implies a primary/combined station) and takes appropriate mode setting action.

The main routines for transmitting and receiving I and supervisory frames, including RR, RNR, REJ and SREJ, are described in the following pages.

SENDI process - Transmits I and all supervisory frames

REFERENCES: P/F received
Information transfer mode
SENT SREJ state variable
P/F PREVIOUS State variable
STATION BUSY state variable
REMOTE BUSY state variable
SREJ ACTIONED state variable

MODIFIES: SNDSREJ flag
SENDRJ return
S - send variable
TIME OUT

CALLS: SENDRJ
SNDRR
SNDRNR

EXIT: NONE

FUNCTION: The function of the SENDI process is to transmit I frames and to transmit supervisory frames as required. The process begins with a determination of whether or not to retransmit a previously transmitted SREJECT. If not, a REJECT or SREJECT may be transmitted depending on the SNDREJ/SNDSREJ flag. Then, ᶦf data is available for transmission and the remote station is not busy, the loop for transmitting I frames is entered at NEXTI. If SREJ has been actioned, the requested frame is set up for retransmission. Next, if P and NRM the F bit is set. SREJ is retransmitted as required. Next the complete I frame is transmitted byte by byte and the send variable (S) is incremented. If the remote station is found to be busy RR or RNR is transmitted as appropriate. If more frames are available for tansmission the loop is repeated from NEXTI. If not, the process is repeated from SENDI.

SNDRR Subroutine - Sends Receive Ready

REFERENCES:
P/F received
Information transfer mode
SENTSREJ state variable
P/F PREVIOUS state variable
STATION BUSY state variable
R - receive variable

MODIFIES:
SNDSREJ state variable
TIMEOUT

CALLS: SENDRJ

EXIT: NORMAL RETURN

FUNCITON: This routine is used to send Receive Ready and also to send SREJECT if appropriate. First, a decision is made whether or not to retransmit a previously sent SREJ. If not, the SENDRJ subroutine is called to send REJ or SREJ depending on the SNDREJ and SNDSREJ flags and the STATION BUSY state variable. Finally, RECEIVE READY is transmitted (or not) depending on the P/F bit and the information transfer state.

SNDRNR Subroutine - Sends Receive Not Ready

REFERENCES: P/F received

Information transfer mode

R - receive variable

MODIFIES: TIMEOUT

EXIT: NORMAL RETURN

FUNCTION: This routine sends Receive Not Ready depending on the P/F bit and the information transfer state. Functionally it is similar to SNDRR, except that there is no provision to send SREJ because RNR Implies that the station is busy.

SENDRJ Subroutine - Transmit REJECT or SREJECT routine

REFERENCES: STATION BUSY state variable
SNDREJ/SNDSREJ flag
R - receive variable

MODIFIES: SNDREJ/SNDSREJ flag
Timeout

EXIT: NORMAL RETURN

FUNCTION: After determining that STATION BUSY is false, and that either SNDREJ or SNDSREJ is true, a supervisory command/response of REJ or SREJ is transmitted, as appropriate. The SNDREJ/SNDSREJ flag is set false before returning.

I Module - Receive I frame routine

REFERENCES: OPERATIONAL STATE Varaible
FRMR - frame reject state variable
LENGTH - number of bytes in information field
P/F received
N(S) received
R - receive variable
REJ/SREJ SENT state variable

MODIFIES: LENGTH
R
REJ/SREJ SENT
SNDREJ/SNDSREJ flag

EXIT: RCV
FRMRI

FUNCTION: This routine is used to make the appropriate checks on the I-frame control field and to read the information field byte-by-byte. First, a check is made to ensure that the receiver is either in information transfer state (ITS) or frame reject state (FRMR). If so, the information field is read in a byte at a time. Next, tests are made for a good frame check sequence (FCS), no FRMR, and an I-frame information field of the correct length. If all of these tests pass, the send sequence number (N(S)) received is compared with the receive variable (R) and appropriate action taken depending on whether or not there is an error and on the method of error recovery. One of the three methods of error recovery that are presented may be used at a given time. These include REJECT recovery, SREJECT recovery, and checkpoint recovery.

SREJECT recovery is used to request that a specific frame be retransmitted. Following frames are buffered until the requested frame is received and then the receive variable is advanced accordingly. REJECT is similar except that all frames following the frame in error must be retransmitted. If neither REJECT nor SREJECT is employed, the checkpointing mechanism must be relied on to perform error recovery: thereceiver simply ignores the frame, and the transmitter must discover the error via N(R) and the P-F checkpoint cycle.

RR/RNR Module - Receive Ready/Receive not Ready

REFERENCES: FRMR - frame reject state variable

MODIFIES: REMOTE BUSY - state variable

CALLS: CHKPNT

EXIT: RCV
FRMRI

FUNCTION After the frame has been validated the CHKPNT routine is called to update the acknowledgement of frames through N(R)-1. If not in frame reject state the REMOTE BUSY state variable is cleared or set depending on whether a receive ready (RR) or receive not ready (RNR) was received.

SREJ Module - Receive SREJECT routine

REFERENCES: N(R) received

P/F received

FRMR - frame reject state variable

REJ/SREJ ACTIONED state variable

CHECKPOINT RETRANS. state variable

P/F (old) value set when SREJ actioned

N(R) (old)

MODIFIES: Ss - specific send variable

REJ/SREJ ACTIONED State variable

REMOTE BUSY state variable

CALLS: CHKPNT

EXIT: RCV

FRMRI

FUNCTION: The SREJ function is similar to the REJ function. After the frame has been validated, a test is made to determine if REJ or SREJ are currently being actioned. If REJ is being actioned, operation is identical to that of the REJ subroutine (no error recovery is performed). If SREJ is being actioned, the P/F and N(R) of the original are compared to the P/F and

N(R) of the current SREJ. If the original P/F is equal to zero for a secondary or primary/ combined station, the next SREJ is not actioned if the P/F=1 and N(R) has the same value and numbering cycle as the first SREJ. If no REJ/SREJ is being actioned and no checkpoint retransmission is in progress a reject exception condition is established. The P/F bit and N(R) are recorded for possible later use as discussed above.

The single frame to be retransmitted is made known to the transmit subroutine and the CHKPNT routine is called to update the acknowledgement of frames through N(R)-1.

REJ Module - Receive REJECT routine

REFERENCES: N(R) received

P/F received

FRMR - frame reject state variable

REJ/SREJ ACTIONED state variable

CHECKPOINT RETRANS. state variable

MODIFIES: S - send variable

TPAK- variables and status bits

REJ/SREJ ACTIONED state variable

REMOTE BUSY state variable

CALLS: CHKPNT

EXIT: RCV

FRMRI

FUNCTION: After the REJ frame has been validated (no FCS error and not in FRMR state) a test is made to determine if a REJECT or SREJECT command/ response is currently being actioned. If not, and if the frame indicated in the N(R) is not being retransmitted due to the checkpointing mechanism, a reject exception condition is established. The send variable, S, is made equal to the N(R) received and the pointers in the transmit buffer are adjusted accordingly. The CHKPNT routine is called to update the acknowledgement of frame through N(R)-1.

CHKPNT Subroutine - Checkpoint recovery routine

REFERENCES: N(R) received

P/F received

FRMR - frame reject state variable

Information transfer state variable

REJ/SREJ ACTIONED - state variable

MODIFIES: S - send variable

TPACK variables add status bits

REJ/SREJ ACTIONED state variable

EXIT: NORMAL RETURN

FRMREJ

FRMR1

FUNCTION: The N(R) received is check for validity. An invalid N(R) is defined as a number that points to an I-frame that has been transmitted previously and acknowledged, or to an I-frame that has not been transmitted and is not the next sequential I-frame pending transmission. If the N(R) is invalid, FRMREJ is called. Otherise the acknowledgement of frames through N(R)-1 is updated via the adjustment of pointers and variables in TPACK. If the poll (or final) bit is set, a check is made to determine if N(R)-1 points to the frame transmitted previously with the final (or poll) bit set. If not, the send variable (S) is reset to the earliest outstanding frame preceding the frame with the final (poll) bit set, as long as this frame is not being transmitted due to REJ or SREJ.

RCNTRL Subroutine - Unpacks control field

REFERENCES: CNTFLD - input control field

COMTAB - command table

VALTAB - command validity table

FRTAB - frame type table

STATE - station type and mode

MODIFIES: POLLP - provisional poll/final bit

NRP - provisional N(R)

NSP - provisional N(S)

FTYPE - frame type

EXIT: NORMAL RETURN

FUNCTION: A provisional P/F bit is extracted from the control field. Next, the frame type is determined by matching the masked control field against a table of implemented commands and responses The I frame is expected first, followed by the supervisory and the unnumbered frames. If no table entry matches the frame type, the command/response is marked invalid. Otherwise the frame type is checked against current station type and mode. If valid, the provisional N(R) and N(S) are extracted and the frame type is returned.

4.3 DETAILED FLOW CHARTS

The detailed flow charts are shown in Figures 4-6 through 4-26. Note that Figure 4-13, Receive I Subroutine, contains three optional recovery routines depending on the method to be used for error recovery. The three routines provide for a pure REJECT recovery, or a pure SREJECT recovery, or neither of these two (checkpoint recovery by default).

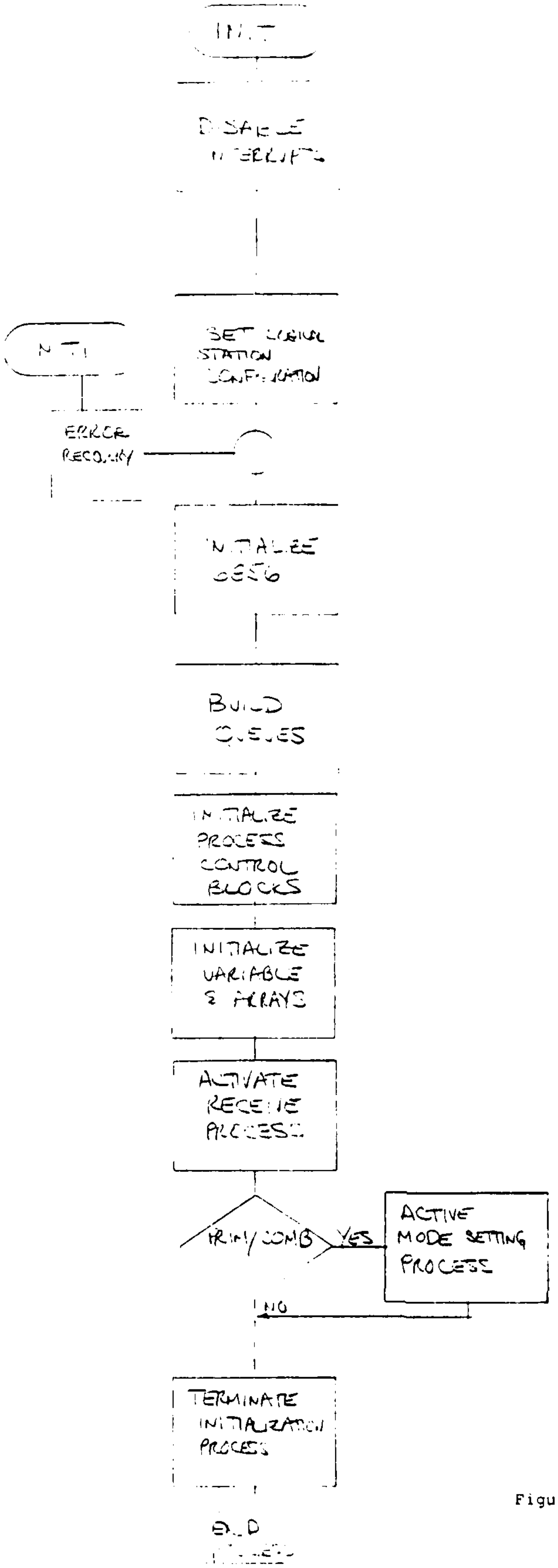
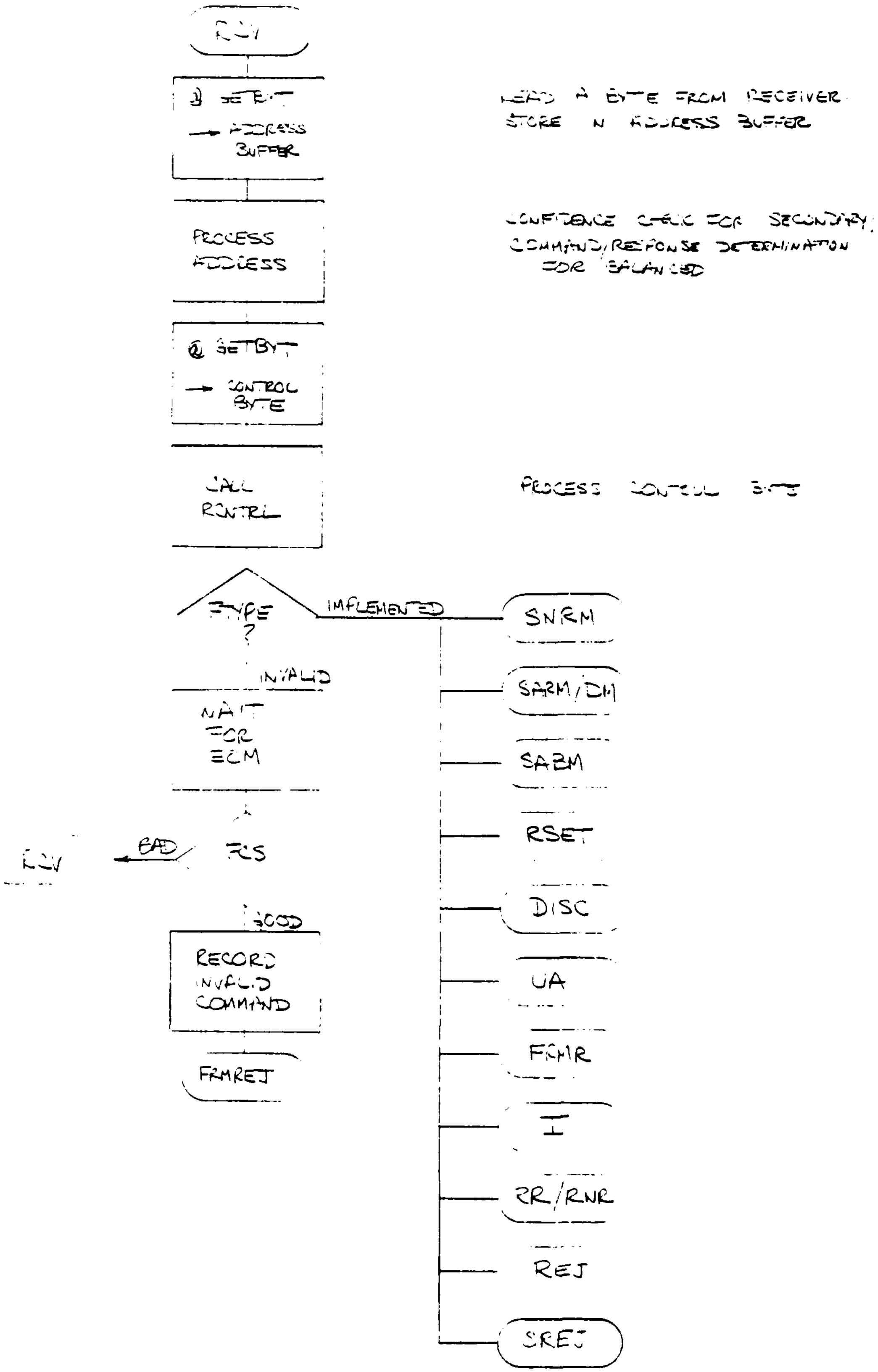

INIT
DISABLE INTERRUPTS
SET LOGICAL STATION CONFIGURATION
INIT
ERROR RECOVERY
INITIALIZE 8256
BUILD QUEUES
INITIALIZE PROCESS CONTROL BLOCKS
INITIALIZE VARIABLES & ARRAYS
ACTIVATE RECEIVE PROCESS
PRIM/COMB
YES
ACTIVE MODE SETTING PROCESS
NO
TERMINATE INITIALIZATION PROCESS
END

Figure 4-6 INIT Process



READ A BYTE FROM RECEIVER
STORE IN ADDRESS BUFFER

CONFIDENCE CHECK FOR SECONDARY;
COMMAND/RESPONSE DETERMINATION
FOR BALANCED

PROCESS CONTROL BYTE

Figure 4-7 RCV Process

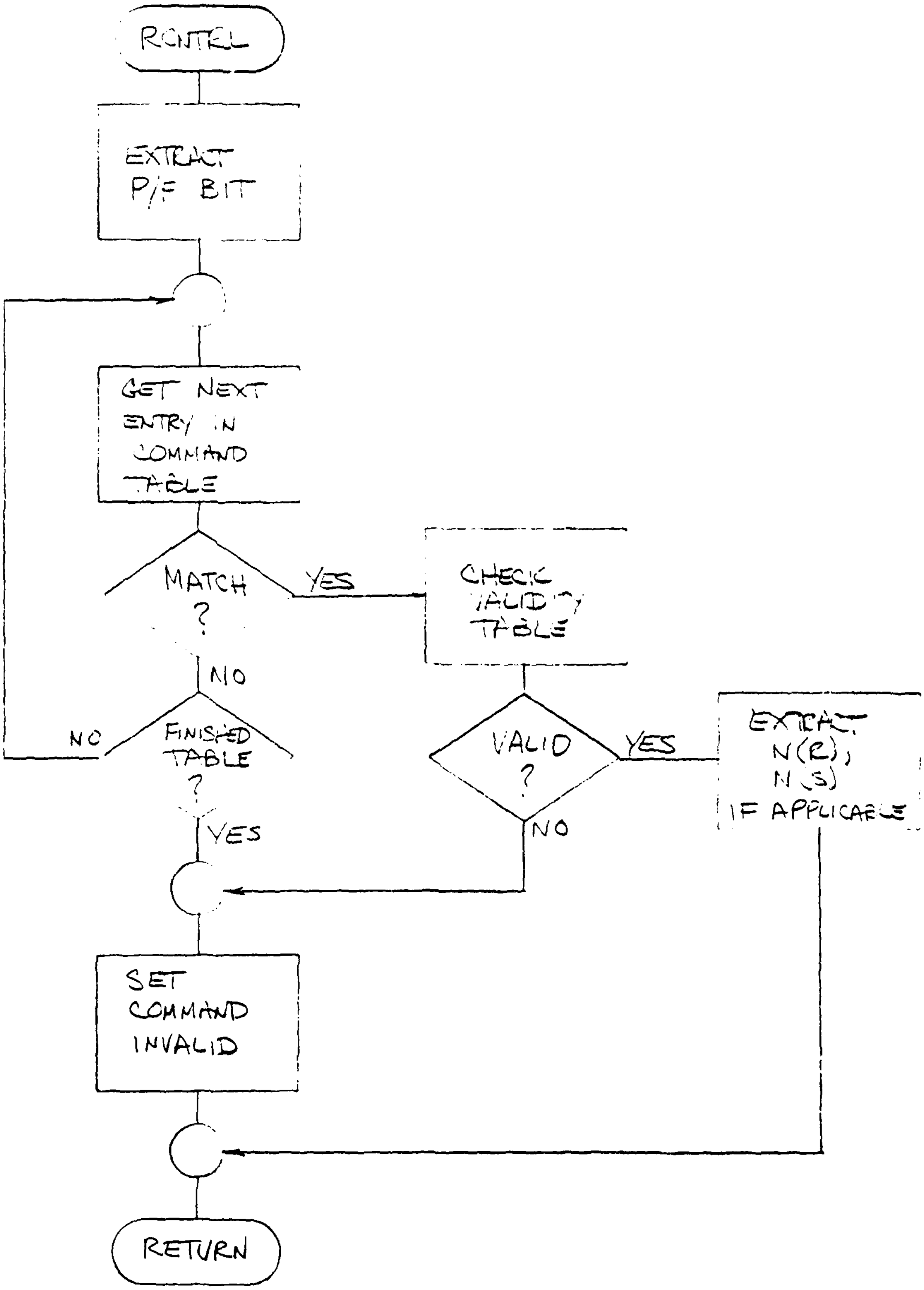


Figure 4-8 RCNTRL Subroutine

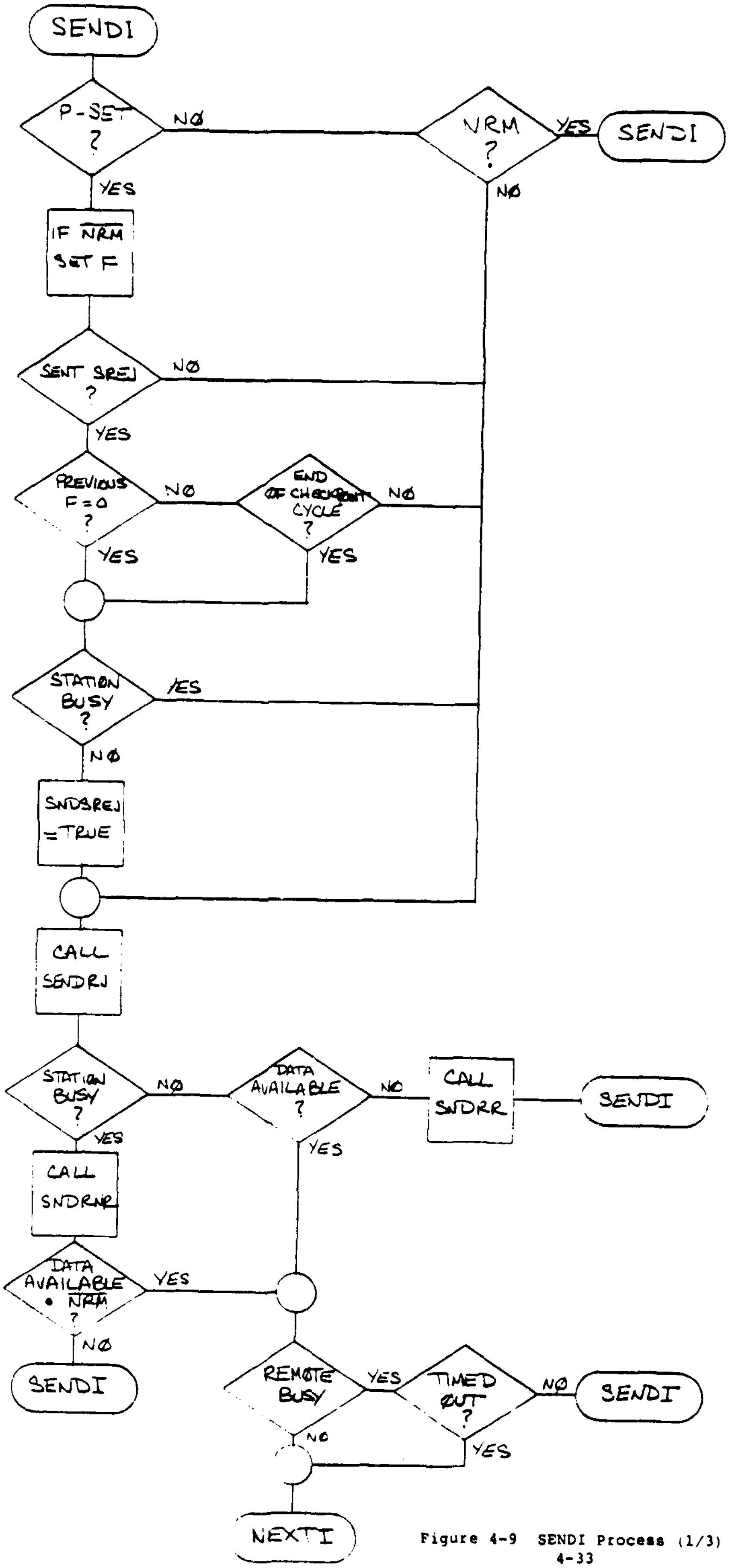


Figure 4-9 SENDI Process (1/3)

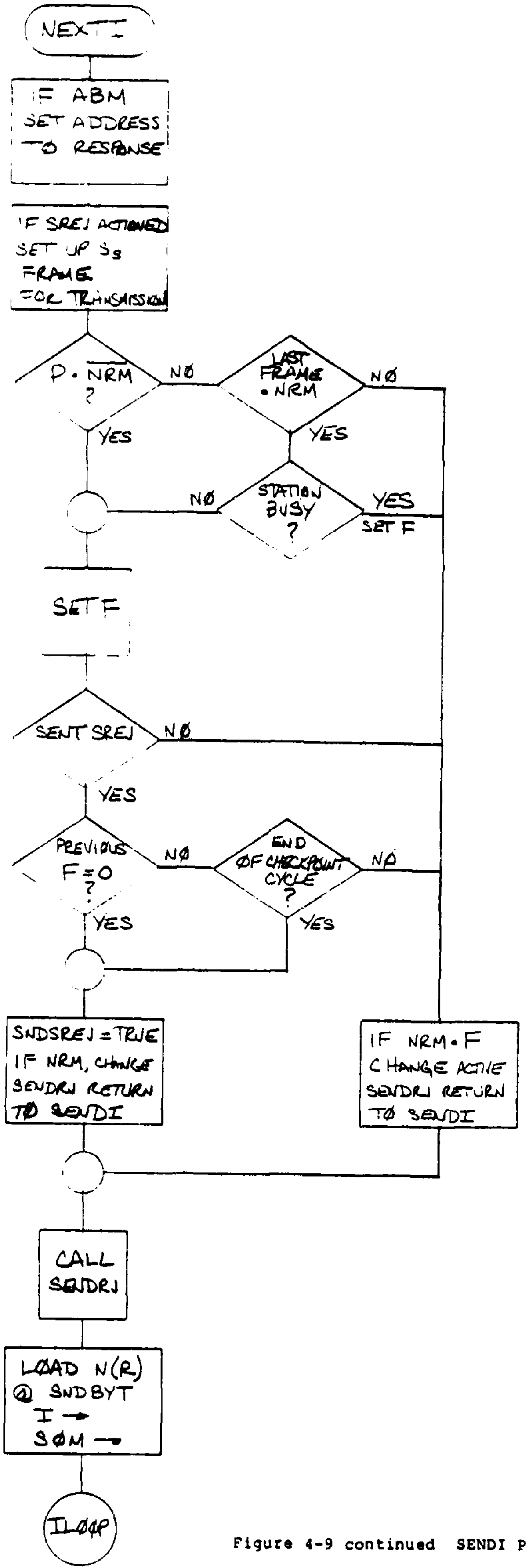

NEXTI
IF ABM SET ADDRESS TO RESPONSE
IF SREJ ACTIONED SET UP S_S FRAME FOR TRANSMISSION
P • NRM ?
NØ
YES
LAST FRAME • NRM
NØ
YES
NØ
STATION BUSY ?
YES
SET F
SET F
SENT SREJ
NØ
YES
PREVIOUS F=0 ?
NØ
END ØF CHECKPOINT CYCLE ?
NØ
YES
YES
SNDSREJ = TRUE IF NRM, CHANGE SENDRJ RETURN TØ SENDI
IF NRM • F CHANGE ACTIVE SENDRJ RETURN TØ SENDI
CALL SENDRJ
LØAD N(R) @ SNDBYT I → SØM →
ILØØP

Figure 4-9 continued SENDI Process (2/3)

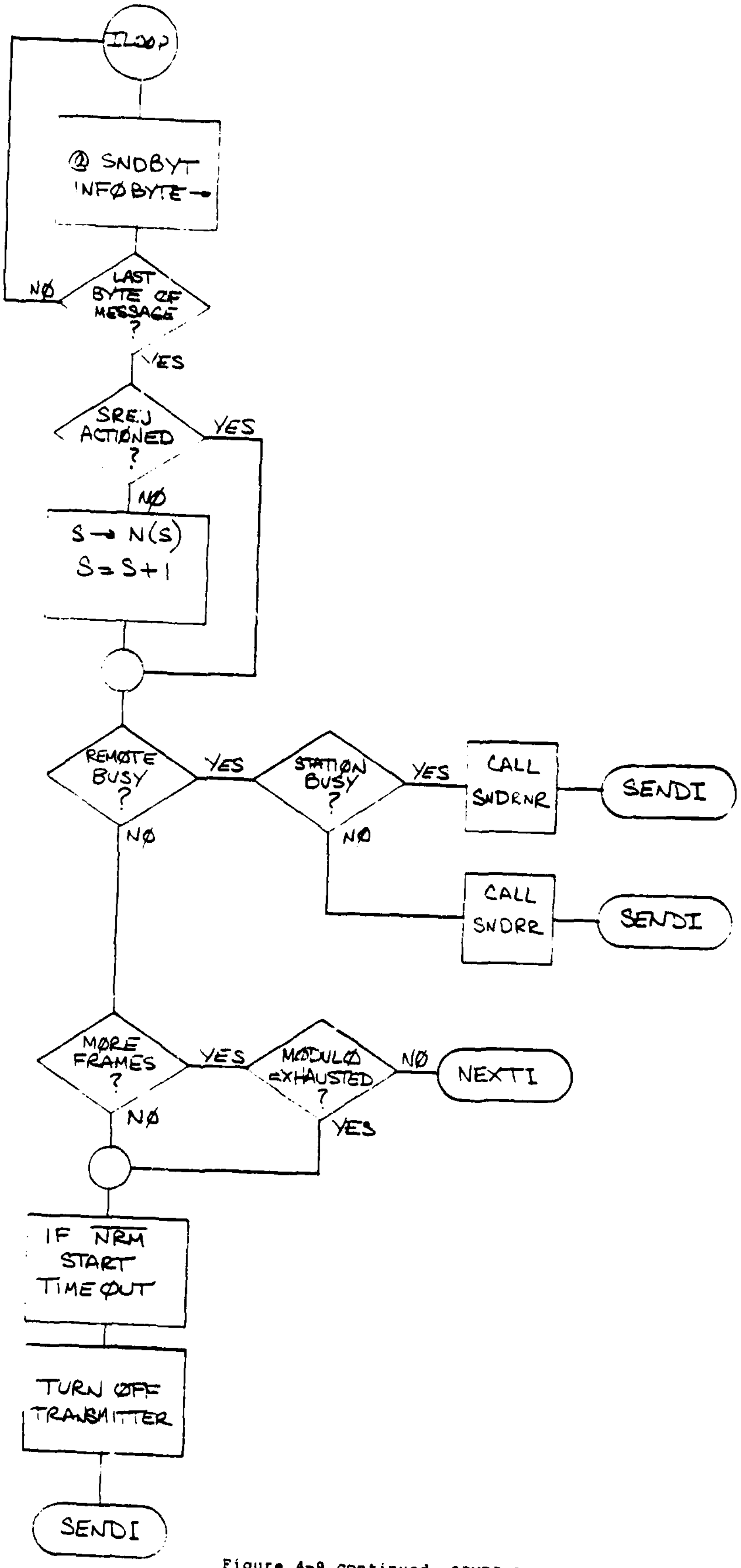


Figure 4-9 continued SENDI Process (3/3)

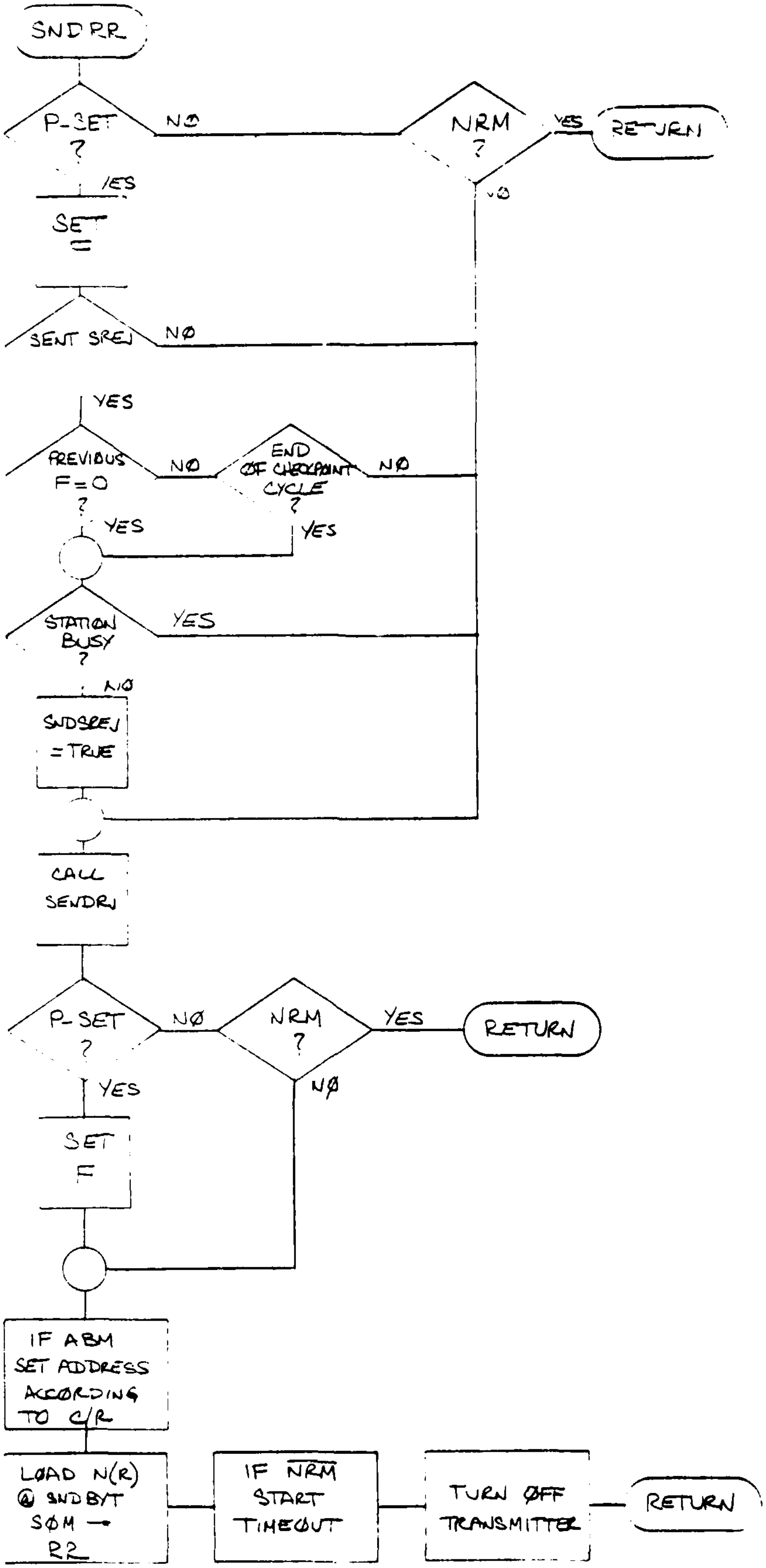


Figure 4-10 SNDRR Subroutine

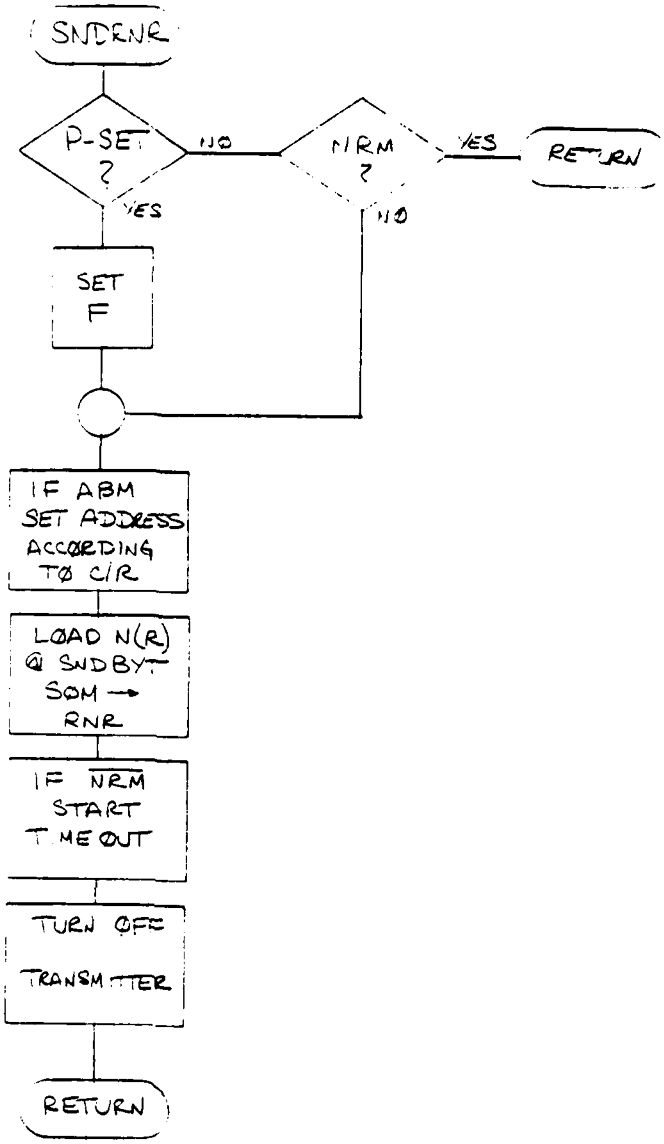


Figure 4-11 SNDRNR Subroutine

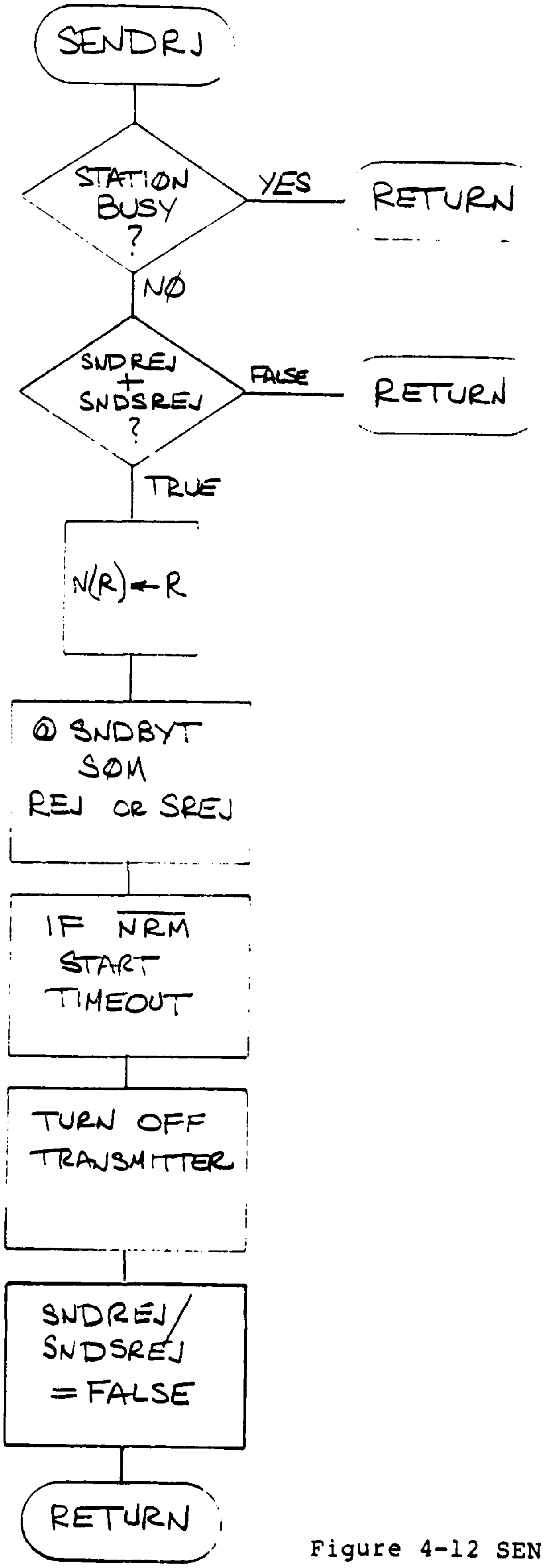


Figure 4-12 SENDRJ Subroutine

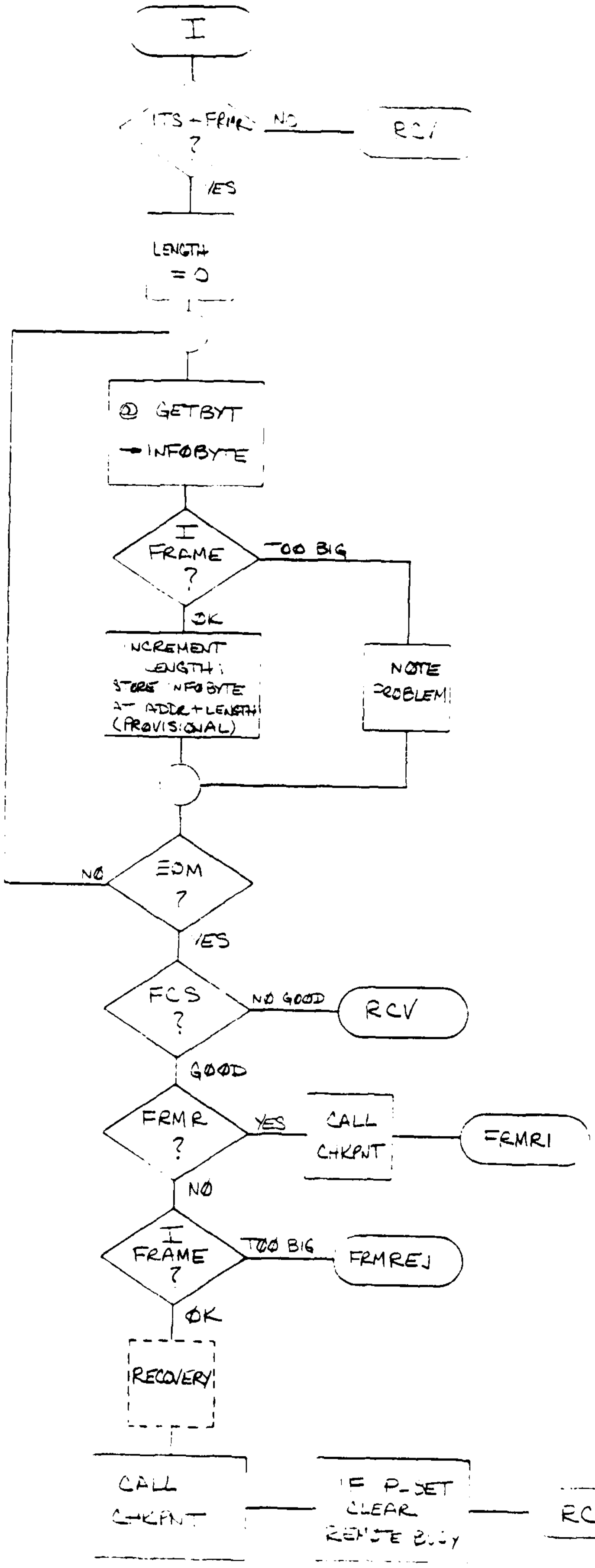
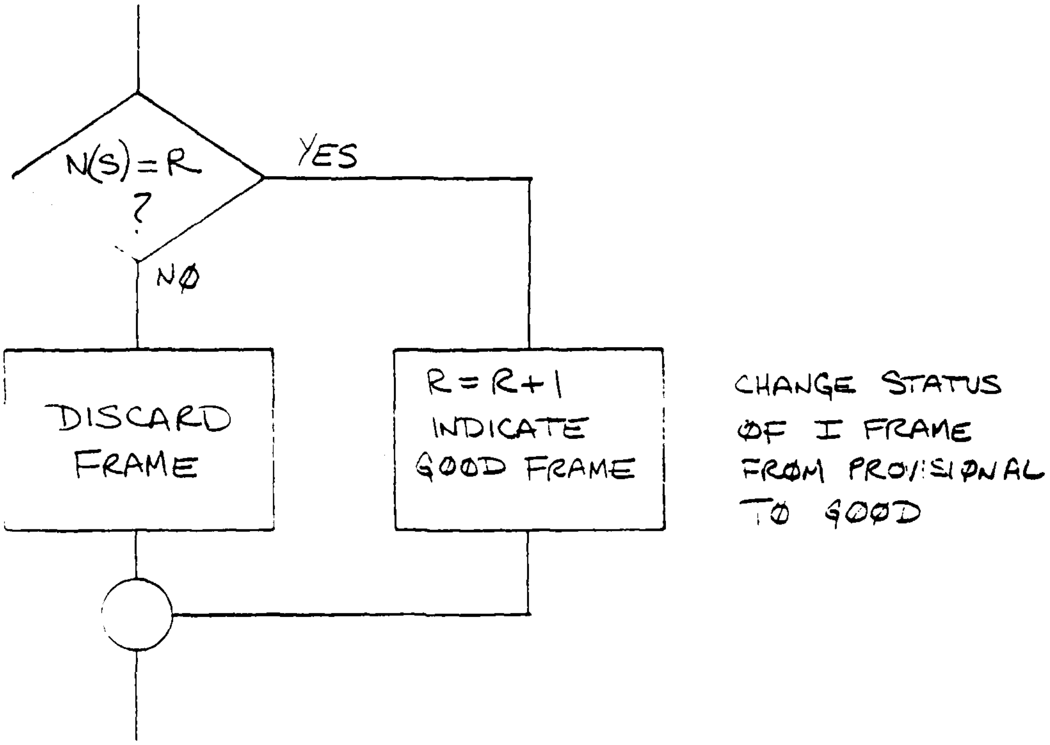


Figure 4-13
Transmit I
Subroutine (1/4)

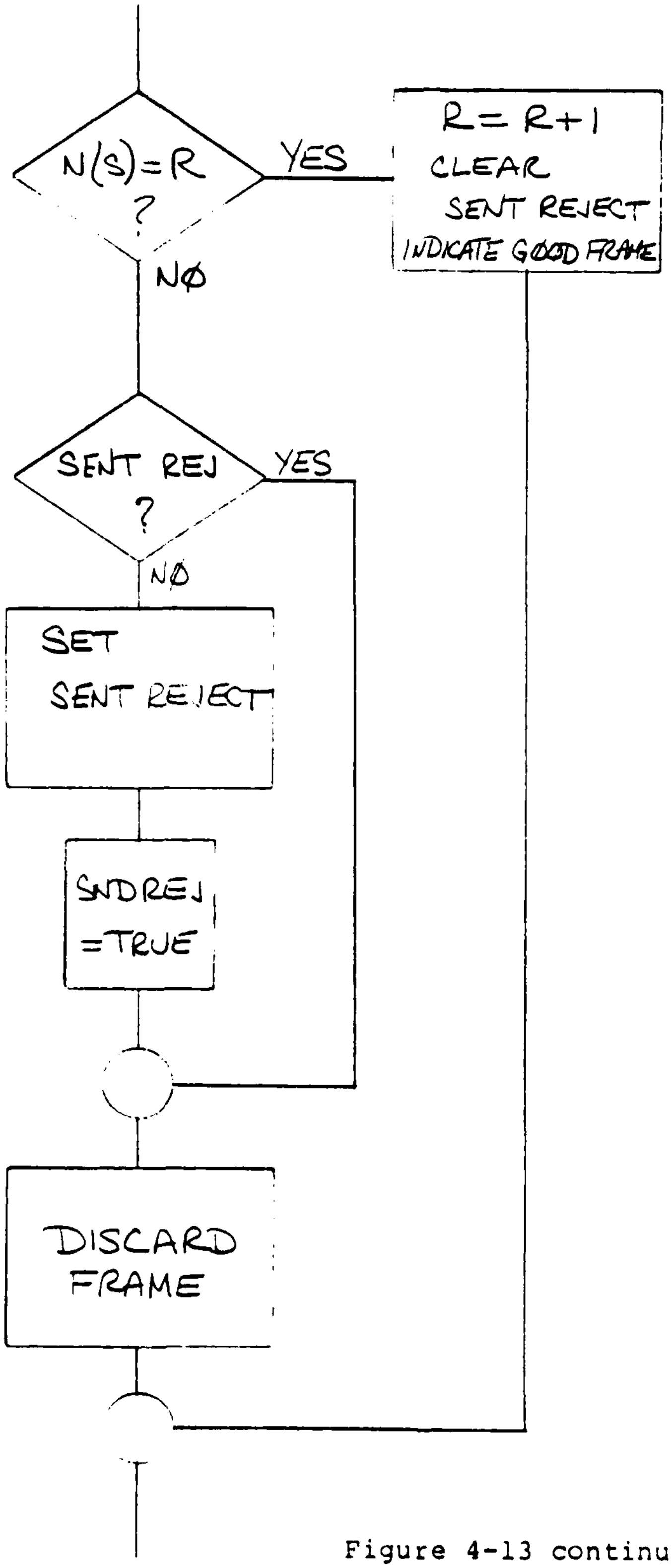
CHECKPØINT RECOVERY



N(S)=R ?
YES
NØ
DISCARD FRAME
R=R+1 INDICATE GOOD FRAME
CHANGE STATUS ØF I FRAME FRØM PROVISIØNAL TØ GØØD

Figure 4-13 continued
Optional Checkpoint Recovery Module
for I Subroutine (2/4)

REJECT RECOVERY



CLEAR REJECT EXCEPTIØN CØNDITION

ESTABLISH REJECT EXCEPTION CONDITION

ENABLE REJ COMM, RESP TØ BE SENT VIA SENDI PROCESS

Figure 4-13 continued
Optional REJECT RECOVERY Module
for I Subroutine (3/4)

SREJECT RECOVERY

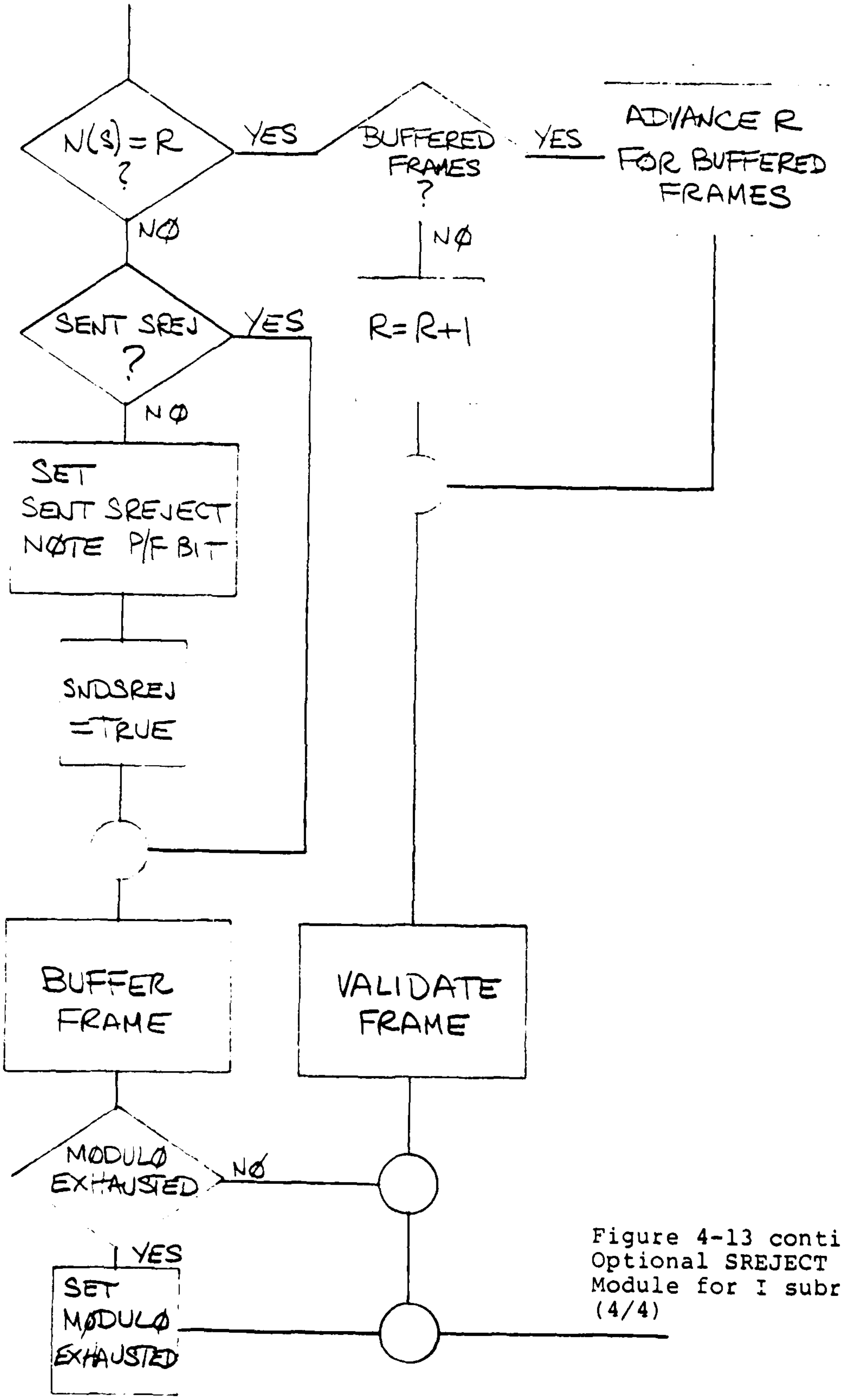


Figure 4-13 continued
Optional SREJECT RECOVERY
Module for I subroutine
(4/4)

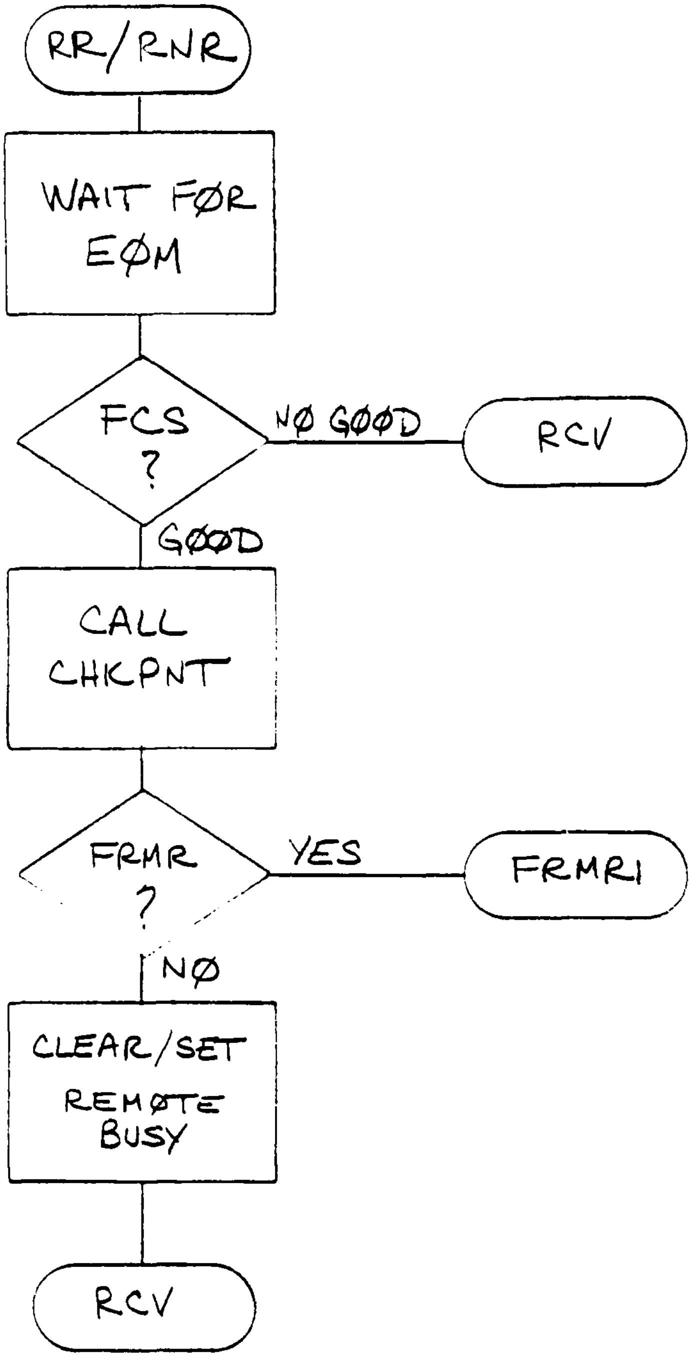
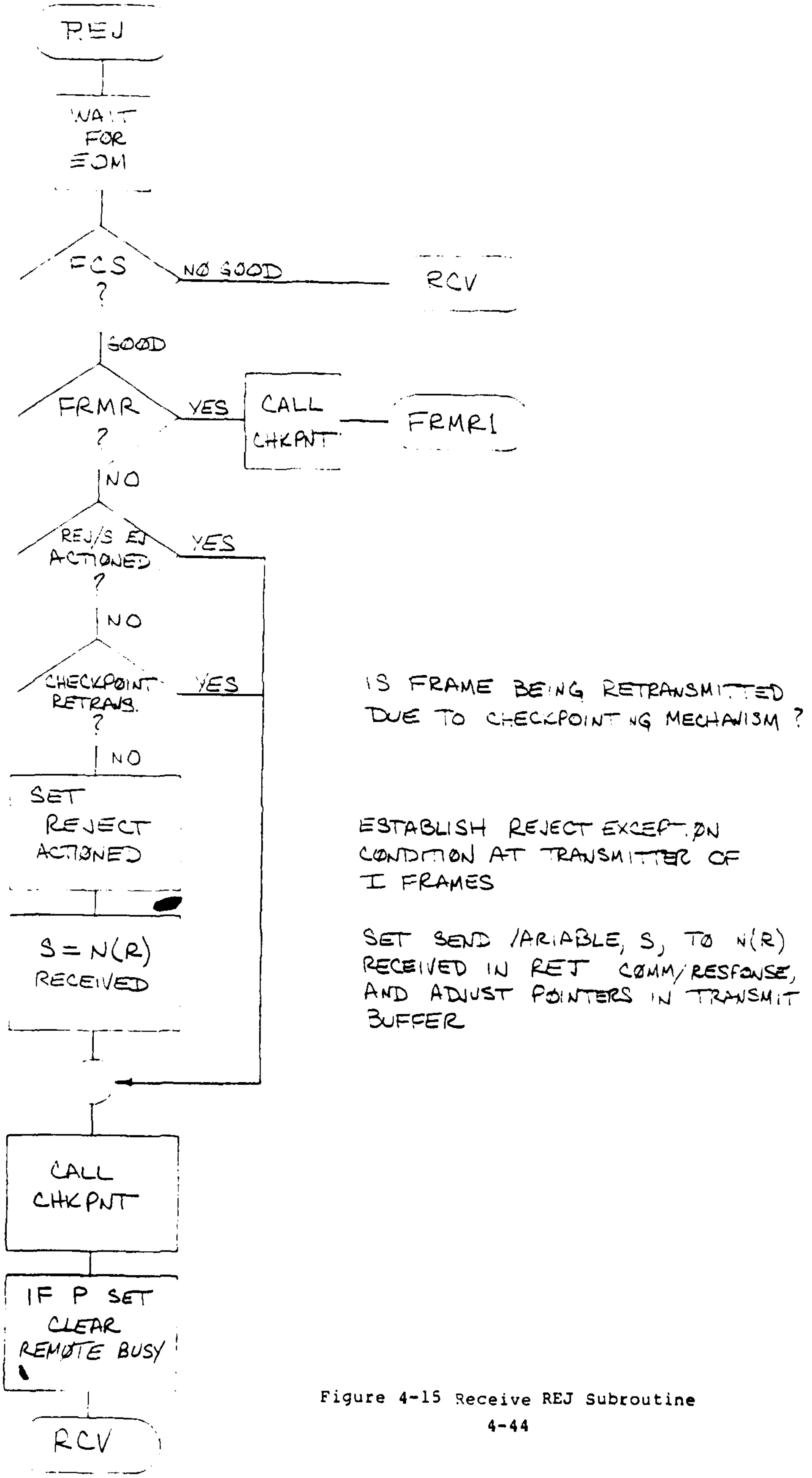


Figure 4-14 Receive Ready/Receive Not Ready Subroutine

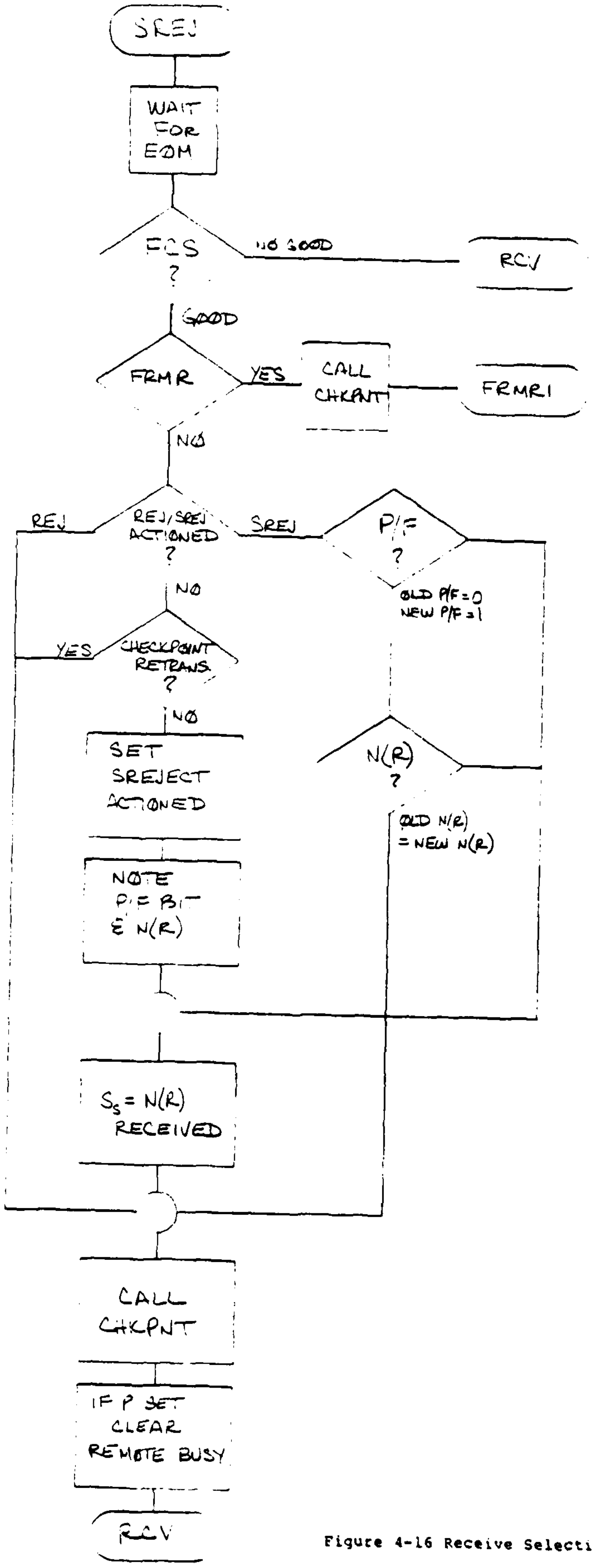


IS FRAME BEING RETRANSMITTED DUE TO CHECKPOINTING MECHANISM ?

ESTABLISH REJECT EXCEPTØN CONDITION AT TRANSMITTER OF I FRAMES

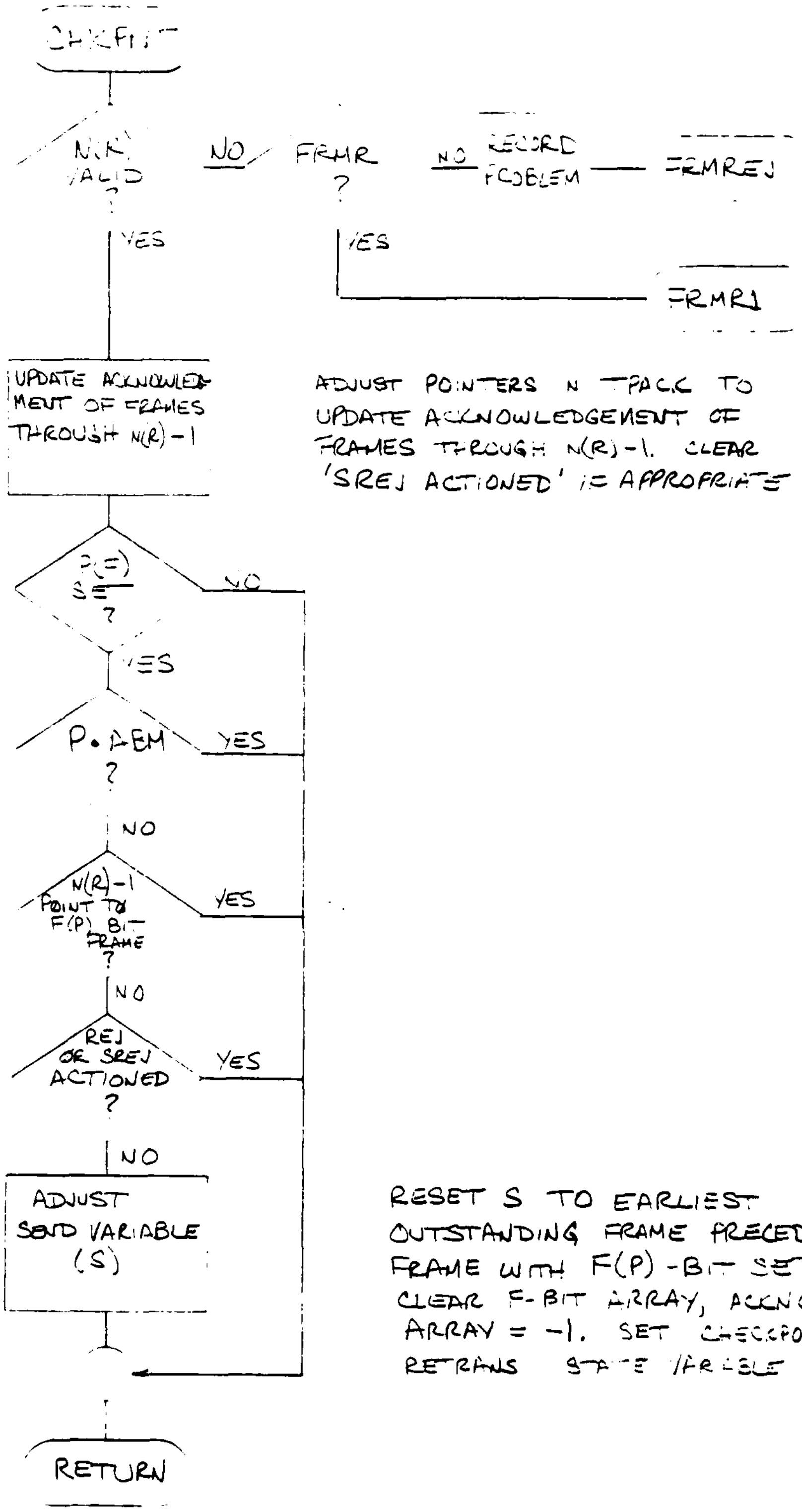
SET SEND VARIABLE, S, TO N(R) RECEIVED IN REJ COMM/RESPONSE, AND ADJUST POINTERS IN TRANSMIT BUFFER

Figure 4-15 Receive REJ Subroutine



IF ORIGINAL P/F = 0 FOR SECONDARY OR PRIMARY/COMBINED, DISABLE ACTIONING NEXT SREJ IF P/F = 1 AND N(R) HAS SAME VALUE AND NUMBERING CYCLE AS THE FIRST SREJ

Figure 4-16 Receive Selective Reject Subroutine



ADJUST POINTERS N TPACK TO UPDATE ACKNOWLEDGEMENT OF FRAMES THROUGH N(R)-1. CLEAR 'SREJ ACTIONED' IF APPROPRIATE

RESET S TO EARLIEST OUTSTANDING FRAME PRECEDING FRAME WITH F(P)-BIT SET, CLEAR F-BIT ARRAY, ACKNOW ARRAY = -1. SET CHECKPOINT RETRANS STATE /ENABLE

Figure 4-17 Checkpoint Subroutine

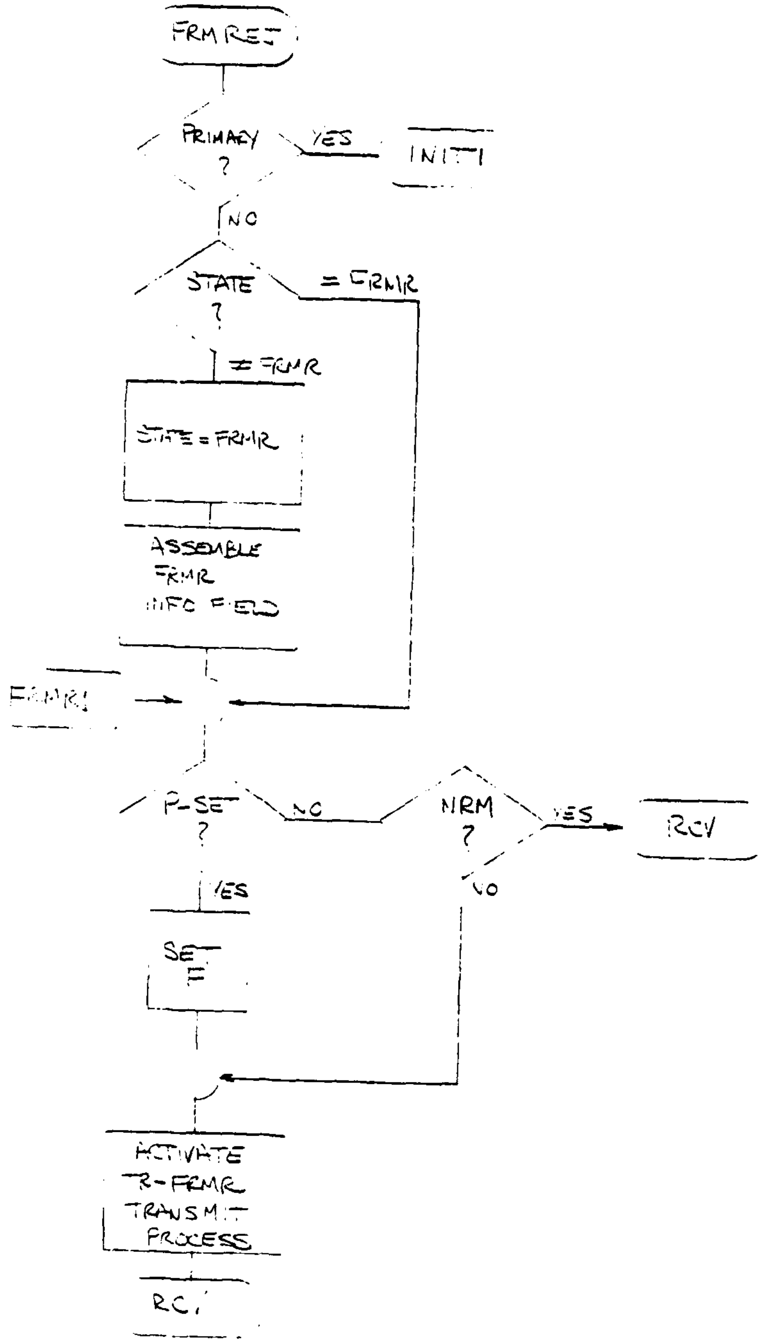


Figure 4-18 Transmit Frame Reject Routine

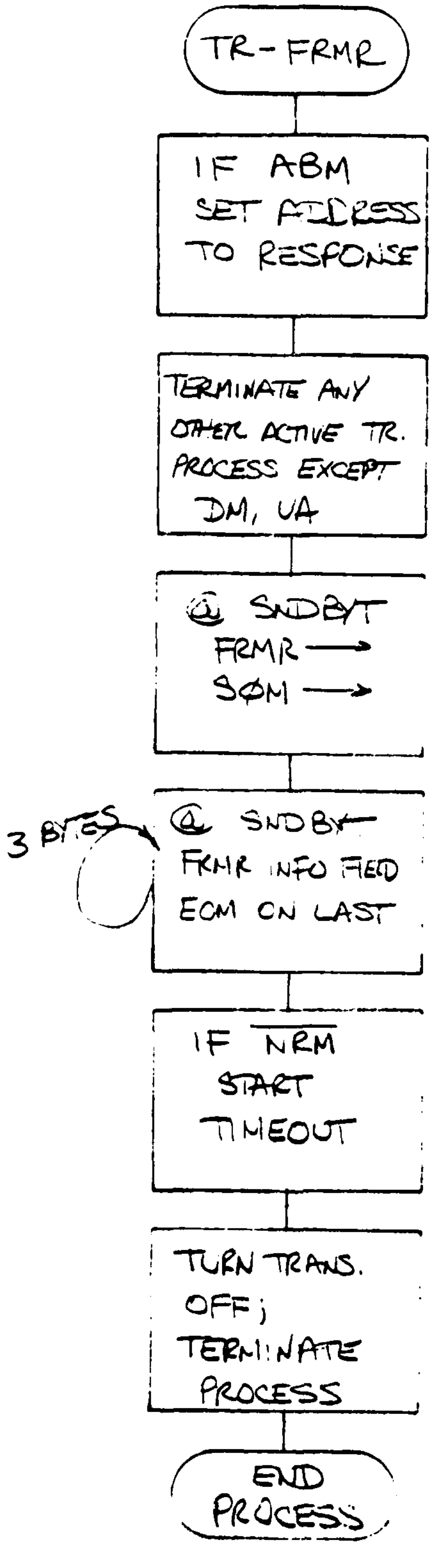

TR-FRMR
IF ABM SET ADDRESS TO RESPONSE
TERMINATE ANY OTHER ACTIVE TR. PROCESS EXCEPT DM, UA
@ SNDBYT
FRMR →
SØM →
3 BYTES
@ SNDBYT
FRMR INFO FIED
EOM ON LAST
IF $\overline{NRM}$ START TIMEOUT
TURN TRANS. OFF; TERMINATE PROCESS
END PROCESS

Figure 4-19 Transmit FRMR Process

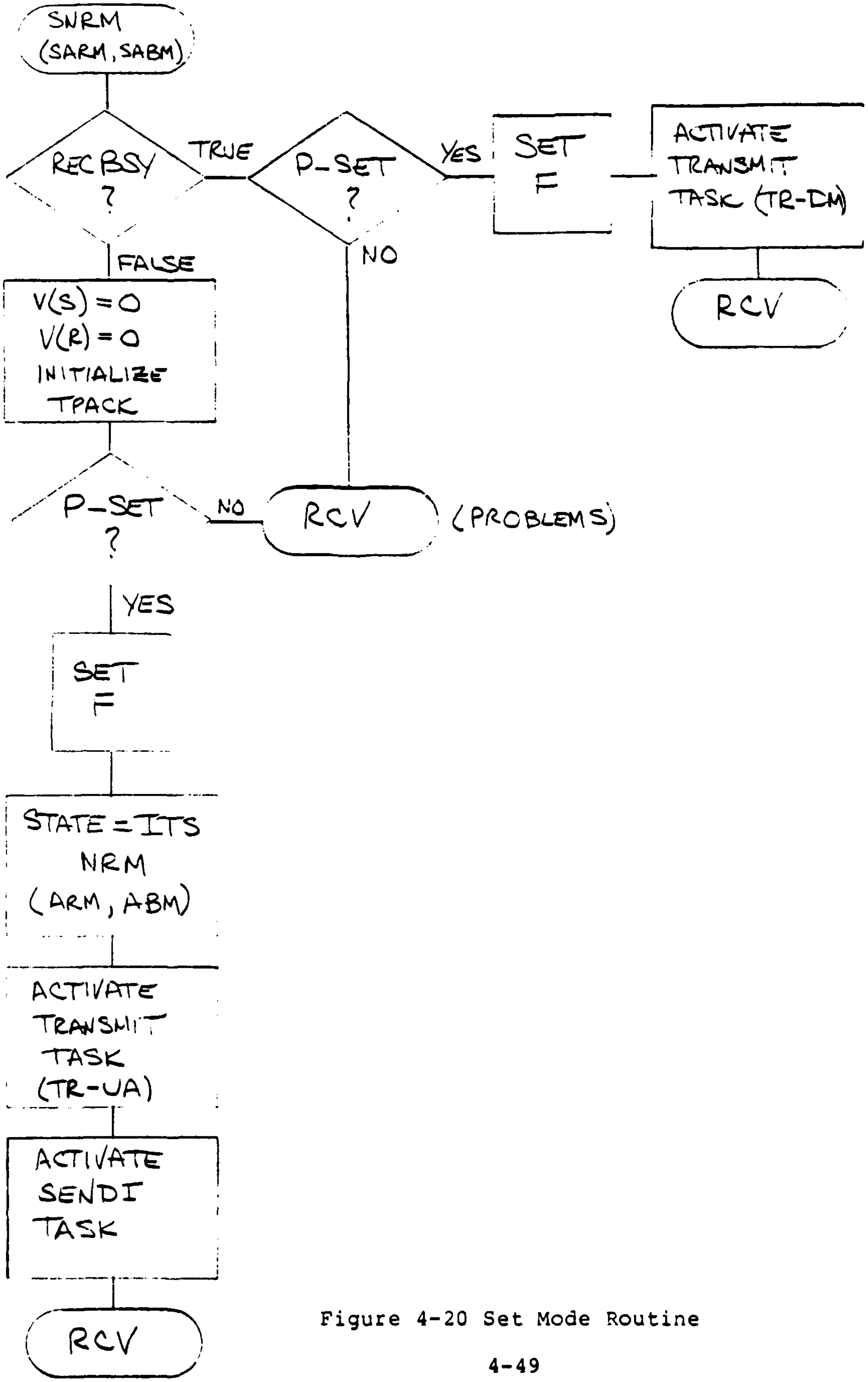


Figure 4-20 Set Mode Routine

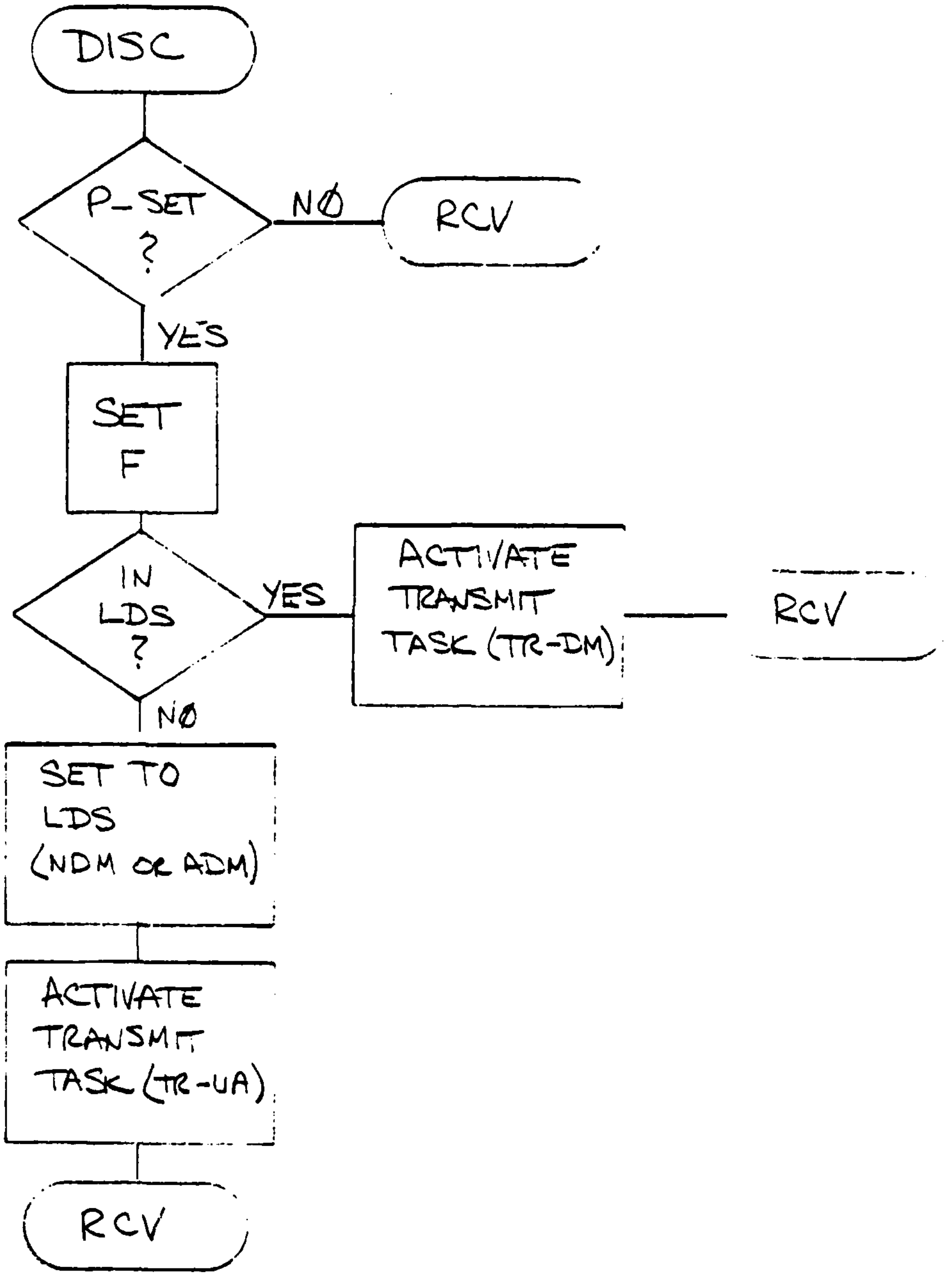
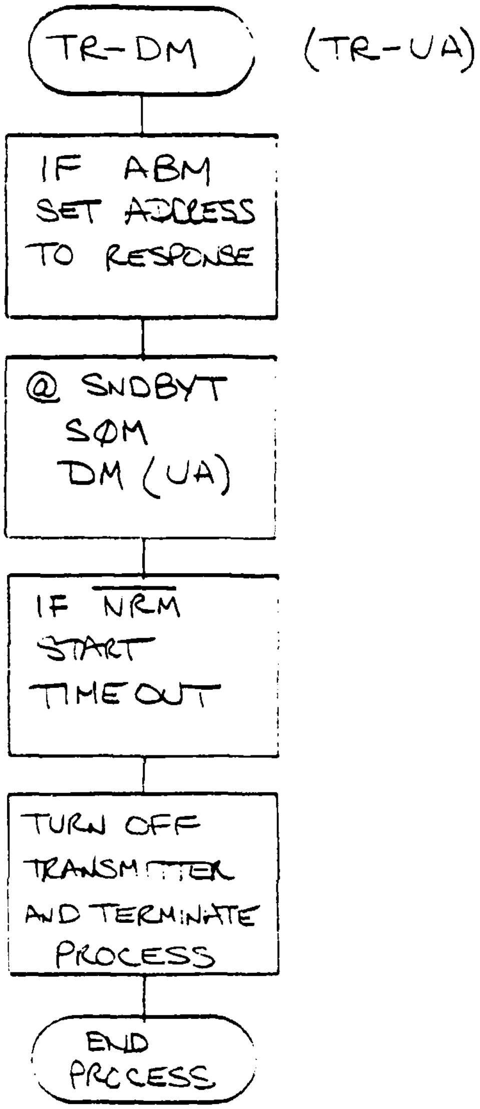


Figure 4-21 Disconnect Routine


TR-DM
(TR-UA)
IF ABM SET ADDRESS TO RESPONSE
@ SNDBYT SØM DM (UA)
IF NRM START TIME OUT
TURN OFF TRANSMITTER AND TERMINATE PROCESS
END PROCESS

IF ABM SET 6356
TO SECONDARY,

TURN ON TRANSMITTER
& SEND DM (OR UA)

Figure 4-22 Transmit DM Process

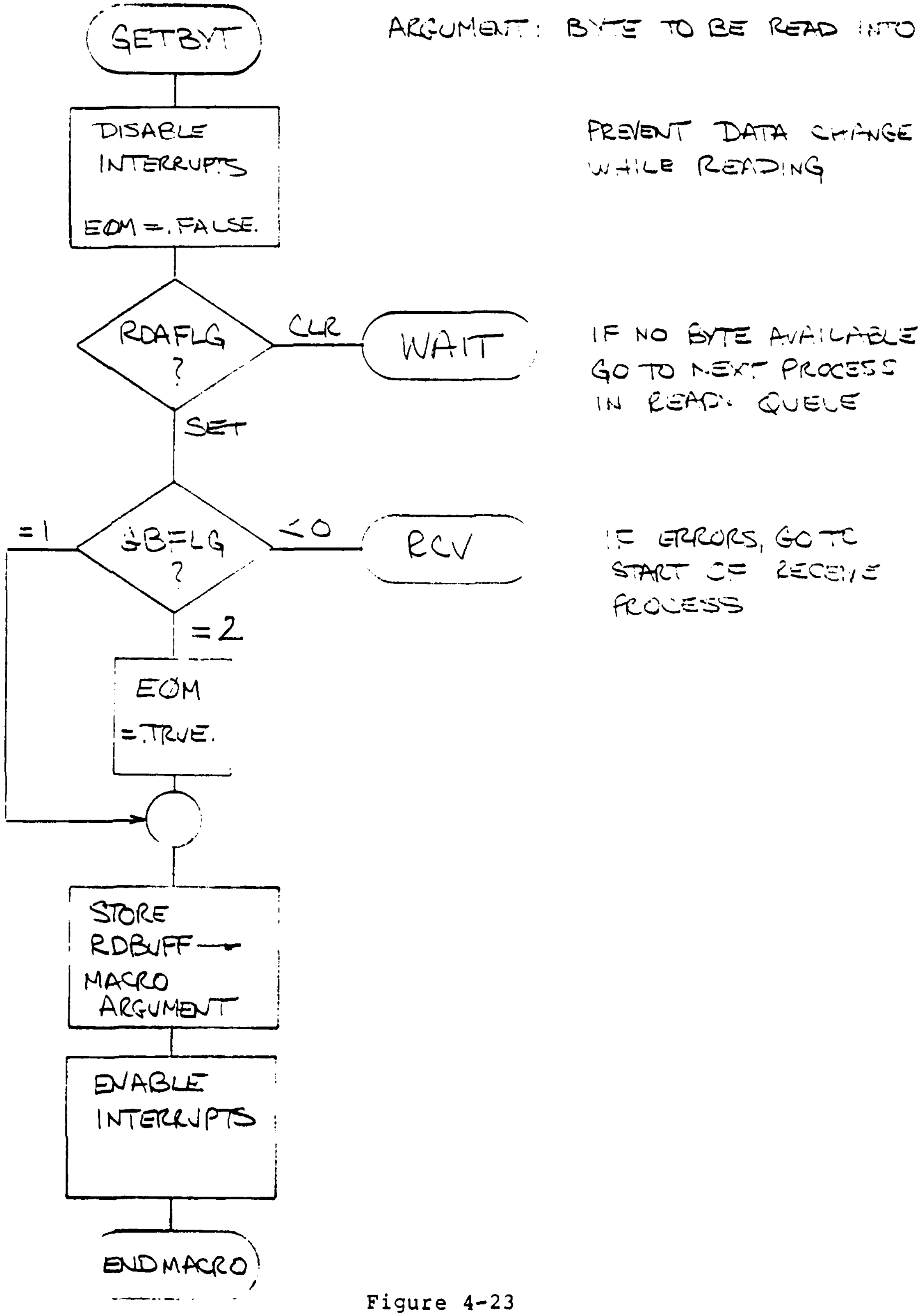
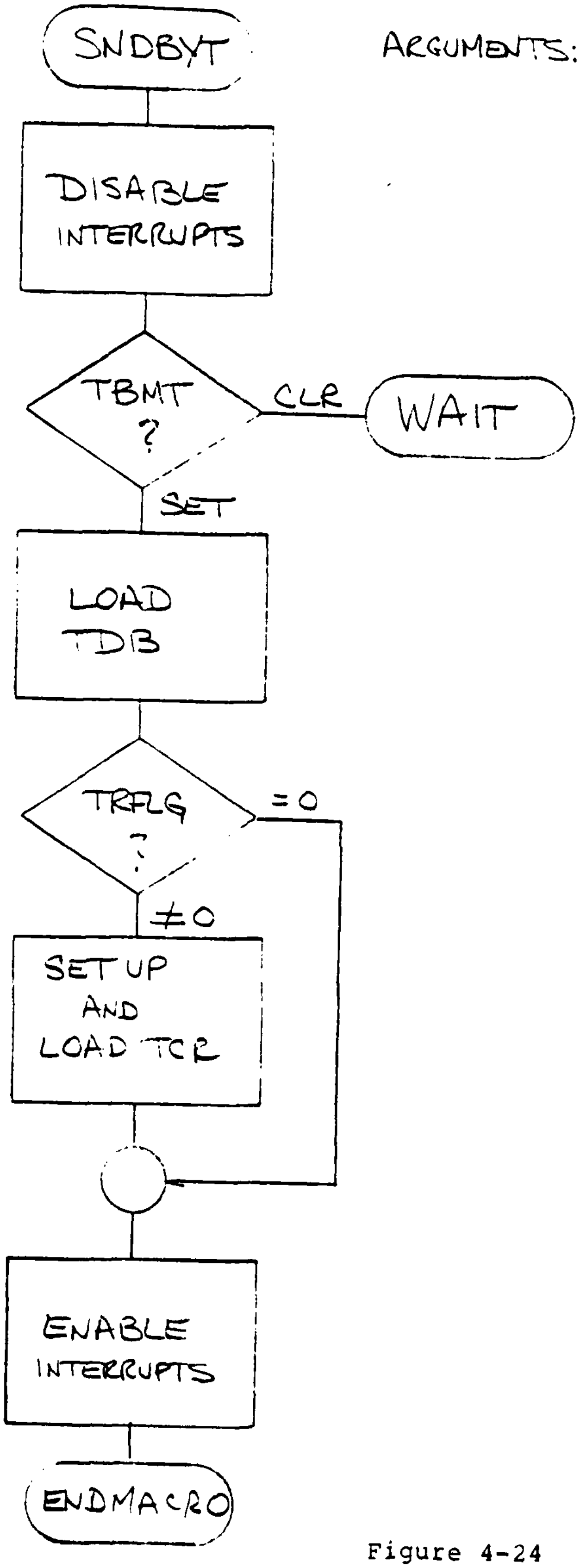

GETBYT
ARGUMENT: BYTE TO BE READ INTO
DISABLE INTERRUPTS
EOM =.FALSE.
PREVENT DATA CHANGE WHILE READING
RDAFLG ?
CLR
WAIT
IF NO BYTE AVAILABLE GO TO NEXT PROCESS IN READY QUEUE
SET
=1
GBFLG ?
<0
RCV
IF ERRORS, GO TO START OF RECEIVE PROCESS
=2
EØM =.TRUE.
STORE RDBUFF → MACRO ARGUMENT
ENABLE INTERRUPTS
ENDMACRO

Figure 4-23
Read Data Byte Macro

ARGUMENTS: BYTE TO BE TRANSMITTED
TRFLG



IF TRANSMITTER BUFFER NOT EMPTY, GO TO NEXT PROCESS IN READY QUEUE

SET UP TCR:
ECM,
SOM,
ABORT

Figure 4-24
Transmit Data Byte Macro

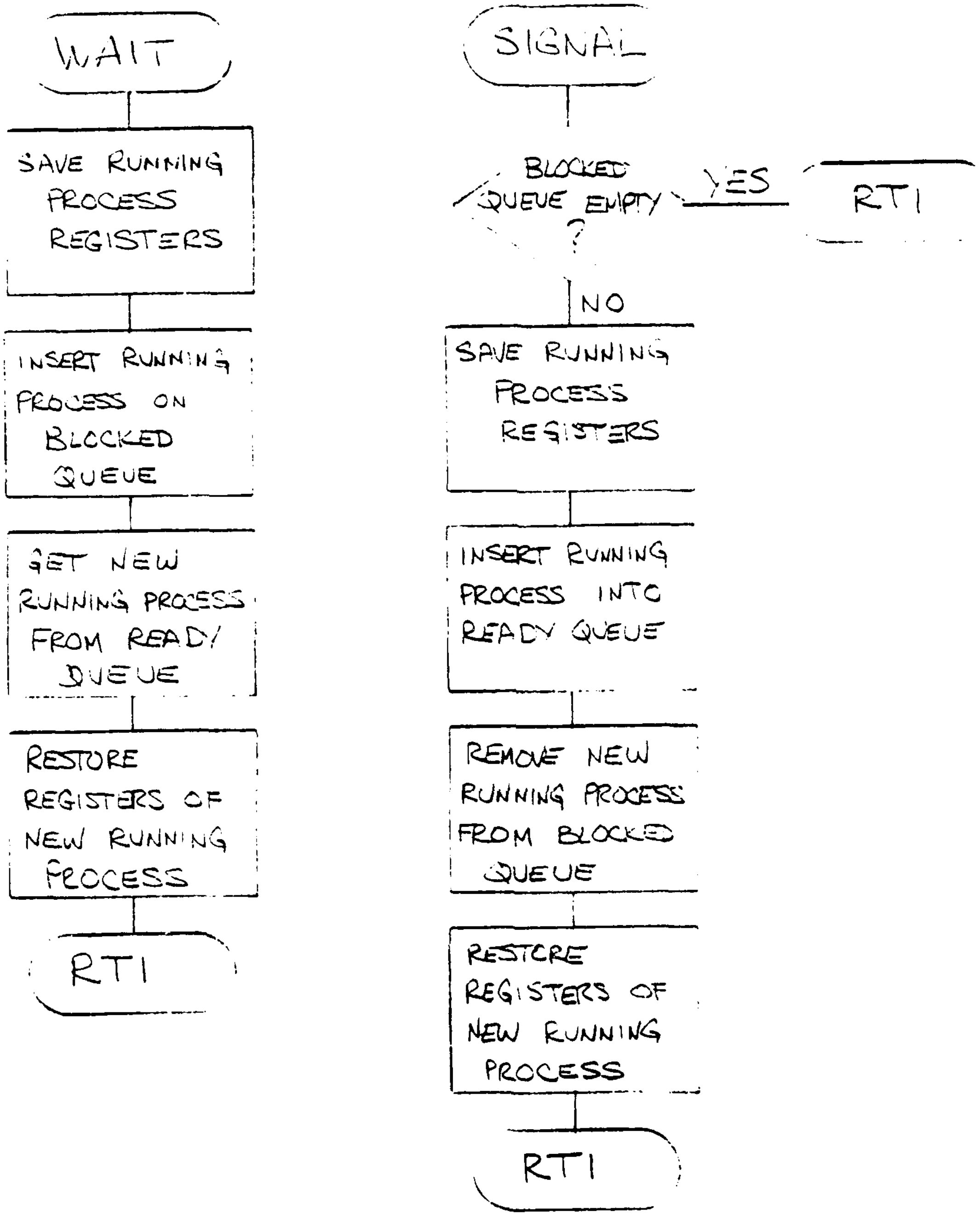

WAIT
SAVE RUNNING PROCESS REGISTERS
INSERT RUNNING PROCESS ON BLOCKED QUEUE
GET NEW RUNNING PROCESS FROM READY QUEUE
RESTORE REGISTERS OF NEW RUNNING PROCESS
RTI
SIGNAL
BLOCKED QUEUE EMPTY ?
YES
RTI
NO
SAVE RUNNING PROCESS REGISTERS
INSERT RUNNING PROCESS INTO READY QUEUE
REMOVE NEW RUNNING PROCESS FROM BLOCKED QUEUE
RESTORE REGISTERS OF NEW RUNNING PROCESS
RTI

Figure 4-25 WAIT & SIGNAL Processes

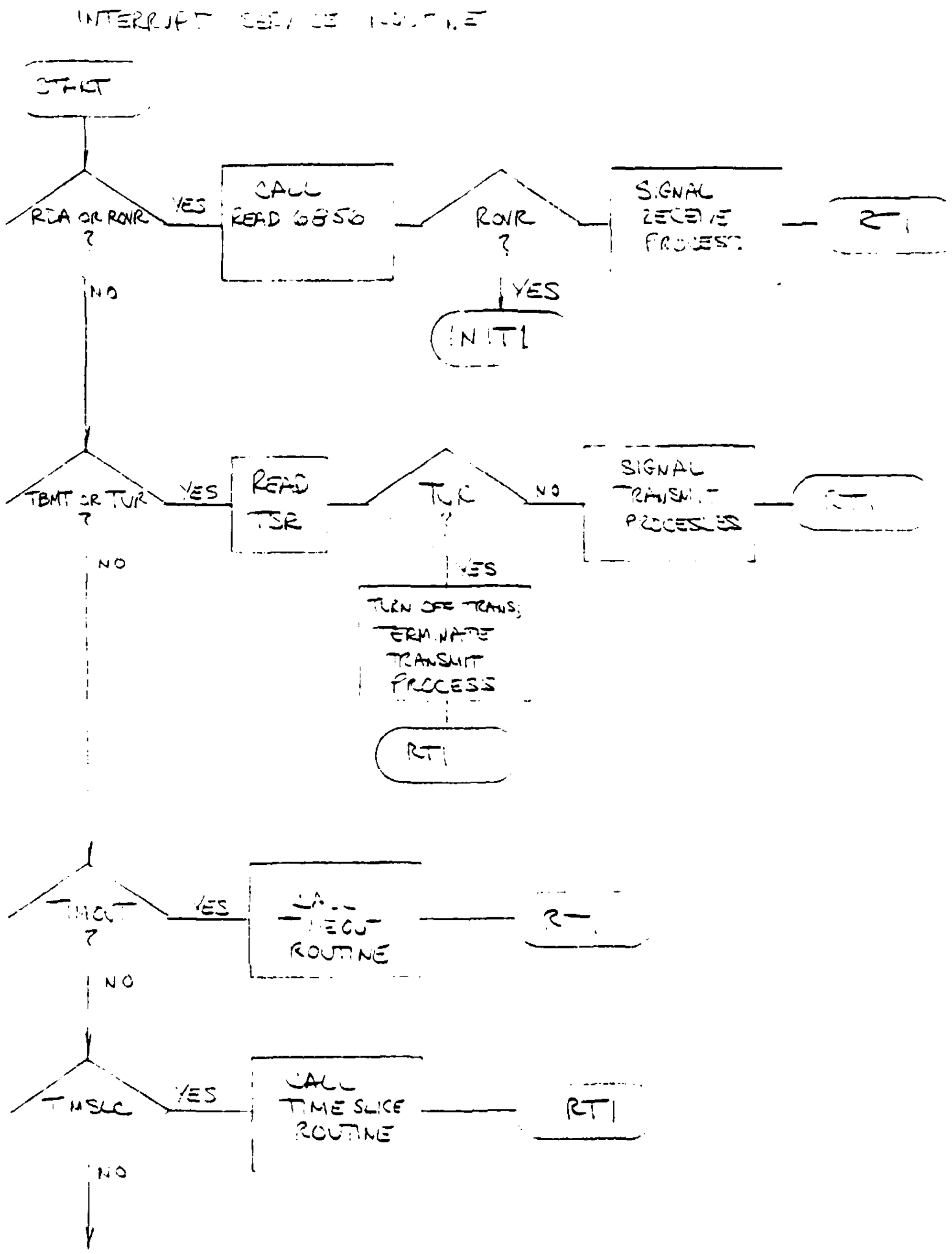


Figure 4-26 Interrupt Service Routine (1/2)

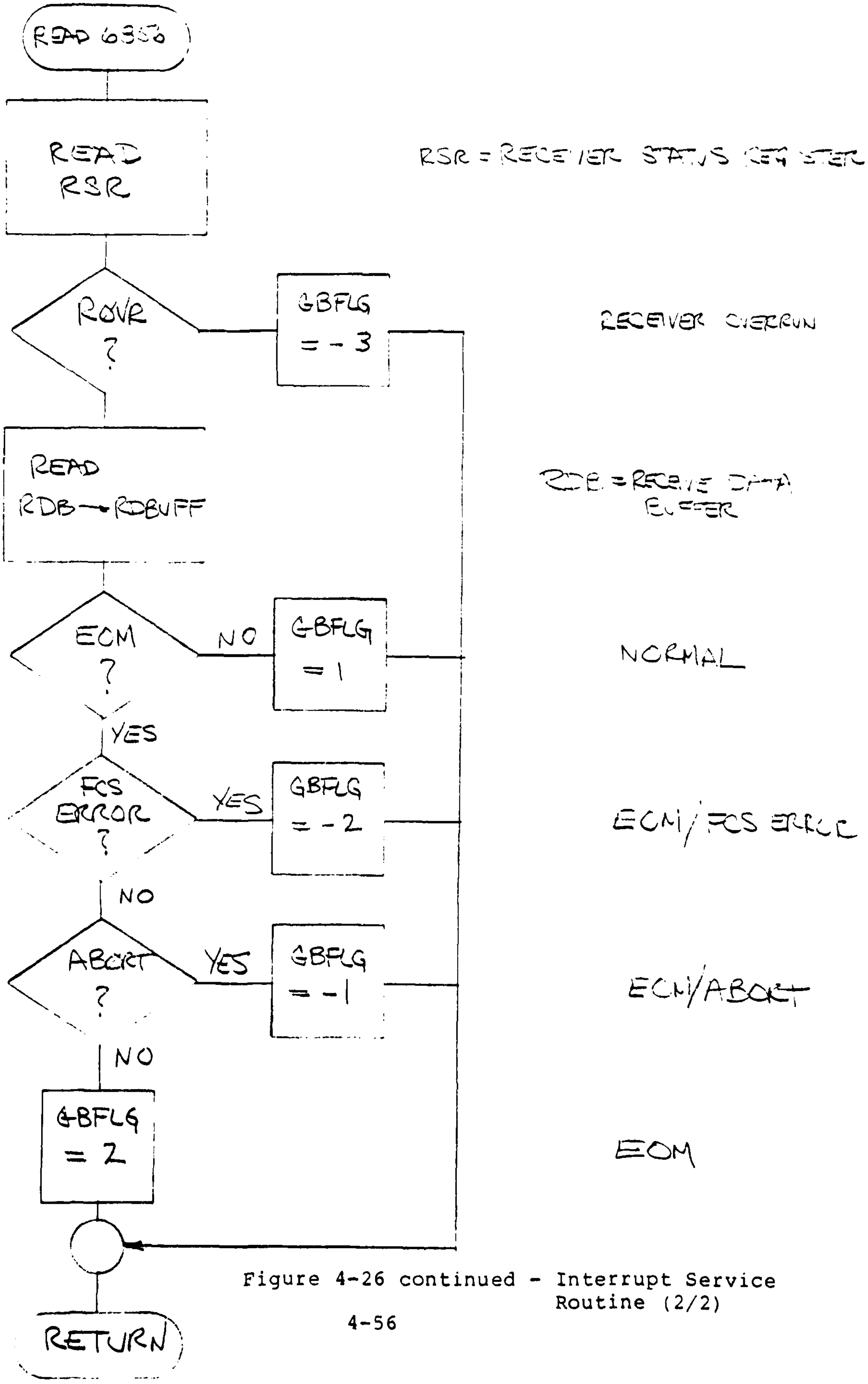


Figure 4-26 continued - Interrupt Service Routine (2/2)

5.0 MICROPROCESSOR CODING AND TESTING

An example of the 6800 microprocessor code is shown in Figure 5-1. This figure shows the assembly language instructions and assembled code for the RCNTRL subroutine. The description of this subroutine was given in Section 4. This subroutine requires 94 bytes for instructions plus 39 bytes for tables for a total of 133 bytes in ROM. Testing of the 6800 code was accomplished on a 6800-based microcomputer. An exhaustive test was made of all 256 possibilities of the input control field. The resulting frame type, P/F bit, N(R), and N(S) extracted was examined for correctness and validity as compared to the frame valid table.

```
ERR LINE  ADDR  B1 B2 B3 B4        TESTR

      30                           *
      31                           *** RCNTRL SUBROUTINE
      32                           *
      33                           *   REFERENCES:  CNTFLD
      34                           *                COMTAB
      35                           *                VALTAB
      36                           *                FRTAB
      37                           *                STAT
      38                           *   MODIFIES:    POLLF
      39                           *                NRP
      40                           *                NSP
      41                           *                FTYPE
      42                           *
      43                           *   EXTRACTS PROVISIONAL P/F BIT, N(R) AND N(S)
      44                           *   IF APPLICABLE. DETERMINES FRAME TYPE. CHECKS
      45                           *   COMMAND/RESPONSE FOR VALIDITY AGAINST MODE,
      46                           *   STATION TYPE.

      48                                   PSCT
      49  000F  B6 00 00      D    RCNTRL  LDAA    CNTFLD
      50  0012  16                         TAB
      51  0013  C4 10                      ANDB    #$10
      52  0015  F7 00 01      D            STAB    POLLF     EXTRACT PROVISIONAL POLL BIT

      54  0018  CE 00 6D      P            LDX     #COMTAB
      55  001B  16                         TAB
      56  001C  C4 01                      ANDB    #$01      TEST FOR I-FRAME FIRST
      57  001E  E1 00                      CMPB    0,X
      58  0020  27 21                      BEQ     MATCH
      59  0022  08                         INX               NOT AN I-FRAME

      61  0023  16                         TAB
      62  0024  C4 0F                      ANDB    #$0F      TEST FOR SUPERVISORY FRAMES
      63  0026  E1 00              TEST1   CMPB    0,X
      64  0028  27 19                      BEQ     MATCH
      65  002A  08                         INX
      66  002B  8C 00 72      P            CPX     #COMTAB+5
      67  002E  26 F6                      BNE     TEST1

      69  0030  16                         TAB               NOT A SUPERVISORY FRAME
      70  0031  C4 EF                      ANDB    #$EF      TEST FOR UNNUMBERED FRAMES
      71  0033  E1 00              TEST2   CMPB    0,X
      72  0035  27 0C                      BEQ     MATCH
      73  0037  08                         INX
      74  0038  8C 00 7A      P            CPX     #COMTAB+13
      75  003B  26 F6                      BNE     TEST2

      77  003D  C6 FF              INVLD   LDAB    #-1       INVALID FRAME TYPE
```

Figure 5-1 RCNTRL Code

```
ERR LINE  ADDR  B1 B2 B3 B4        SOURCE

     78   003F  F7 00 01     D           STAB    FTYPE
     79   0042  39                       RTS

     81   0043  F6 00 05     D   MATCH   LDAB    STAT       GET STATION TYPE & MODE
     82   0046  E4 0D                    ANDB    VALTAB-CONTAB,X   TEST AGAINST FRAME TYPE
     83   0048  27 F3                    BEQ     INVLD
     84   004A  E6 1A                    LDAB    FRTAB-CONTAB,X
     85   004C  F7 00 04     D           STAB    FTYPE

     87   004F  C1 05                    CMPB    #5
     88   0051  2E 19                    BGT     DONE       UNNUMBERED FRAME
     89   0053  16                       TAB
     90   0054  C4 E0                    ANDB    #$E0       GET N(R) PORTION
     91   0056  54                       LSRB
     92   0057  54                       LSRB
     93   0058  54                       LSRB
     94   0059  54                       LSRB
     95   005A  54                       LSRB
     96   005B  F7 00 02     D           STAB    NRP        STORE PROVISIONAL N(R)
     97   005E  F6 00 04     D           LDAB    FTYPE
     98   0061  C1 01                    CMPB    #1
     99   0063  26 07                    BNE     DONE       SUPERVISORY FRAME
    100   0065  16                       TAB
    101   0066  C4 0E                    ANDB    #$0E
    102   0068  54                       LSRB
    103   0069  F7 00 03     D           STAB    NSP        STORE PROVISIONAL N(S)
    104   006C  39               DONE    RTS
    105                                  PSCT
    106   006D  00               CONTAB  FCB     00000000B INFORMATION
    107   006E  01                       FCB     00000001B RECEIVE READY
    108   006F  05                       FCB     00000101B RECEIVE NOT READY
    109   0070  09                       FCB     00001001B REJECT
    110   0071  0D                       FCB     00001101B SELECTIVE REJECT
    111   0072  83                       FCB     10000011B SET NORMAL RESPONSE MODE
    112   0073  63                       FCB     01100011B UNNUMBERED ACKNOWLEDGEMENT
    113   0074  43                       FCB     01000011B DISCONNECT
    114   0075  0F                       FCB     00001111B DISCONNECTED MODE
    115   0076  87                       FCB     10000111B FRAME REJECT
    116   0077  BF                       FCB     10001111B RESET
    117   0078  0F                       FCB     00001111B SET ASYNC RESPONSE MODE
    118   0079  2F                       FCB     00101111B SET ASYNC BAL MODE

    120   007A  F8               VALTAB  FCB     11111000B I
    121   007B  F8                       FCB     11111000B RR
    122   007C  F8                       FCB     11111000B RNR
    123   007D  F8                       FCB     11111000B REJ
    124   007E  F8                       FCB     11111000B SREJ
    125   007F  40                       FCB     01000000B SNRM
    126   0080  A8                       FCB     10101000B DS
    127   0081  58                       FCB     01011000B DISC
    128   0082  B8                       FCB     10111000B FR
```

```
ERR LINE  ADDR  B1 B2 B3 B4        SOURCE

     129  0083  AB                           FCB    10101000B FRMR
     130  0084  08                           FCB    00001000B RSFT
     131  0085  B8                           FCB    10111000B SARM
     132  0086  08                           FCB    00001000B SABM

     134  0087  01 02 03 04        FETAB     FCB    1,2,3,4,5,6,7,8,9,10,11,12,13
     135  008B  05 06 07 08
     136  008F  09 0A 0B 0C
     137  0093  0D

     139  0094                               END

  ASSEMBLER ERRORS =     0
```

```
                                   TESTR

                              CROSS REFERENCE

LABEL      VALUE        REFERENCE

CNTFLD   D 0000           -7    49
COMTAB   F 006D           54    66    74    82    84  -106
DONE     F 006C           88    99  -104
FRTAB    F 0087           84  -134
FTYPE    D 0004          -11    78    85    97
INVLD    F 003D          -77    83
MATCH    F 0043           58    64    72   -81
MEMORY   M 0000            0
NARG       0000            0
NRP      D 0002           -9    96
NSP      D 0003          -10   103
POLLF    D 0001           -8    52
RCNTRL   F 000F           27   -49
TESF1    F 0007          -23    25
STACK    S 0000            0
STAT     D 0005          -12    81
TEST1    F 0026          -63    67
TEST2    F 0033          -71    75
VALTAB   F 007A           82  -120
```

6.0 DISCUSSION OF FEASIBILITY

As discussed previously, one of the objectives of this program is to determine the practicality of using a microprocessor, such as the M6800, to implement the unbalanced normal, unbalanced asynchronous, and balanced asynchronous class of procedures. Two major factors affecting the feasibility are the number of instructions required to implement the protocol, and the time necessary to execute these instructions. The total number of instructions has a significant effect on the cost of developing a processor-based system, and the throughput (or baud-rate over the communication line in this instance) is determined by the execution speed through critical paths on the program. These factors are disucssed below.

6.1 MEMORY REQUIREMENTS

The number of instructions required to implement the protocol including REJ or SREJ can be estimated by examining the detailed flow charts in Section 4. The protocol including pure SREJ is estimated to require 1300 instructions; this represents an increase of 450 instructions over the protocol without SREJ. The pure REJ protocol is estimated at 100 instructions less or 1200 instructions. Approximately 250 instructions will be required for the operating system, depending on desired features. RAM is required for variable storage and data buffers.

6.2 EXECUTION TIME

The speed at which the microprocessor can execute the protocol in real-time depends to a large extent on the actual hardware/software design: The hardware design can be "standard" or it can include many processes accomplished in hardware (such as the F6856). For the purpose of this program, the standard approach with the aid of the F6856 is assumed. The software design must address the time-critical portions of the simultaneous transmit/receive processes to ensure that these critical processes may be serviced in real time. For this program, no attempt has been made to optimize these processes, since a thorough analysis is required to determine just what is "critical". However, some rough estimates can be made based on the current state of the design.

Assuming a MPU rate of 1 cycle/microsec. it appears that a 9.6 or 19.2 kilobit/sec. transmission rate would not be too difficult. That is, a 19.2 kilobit/sec. rate is equivalent to 400 microseconds per byte transmitted, which is approximately 100 instructions. It should be possible to implement the critical parts of the send/receive process using between 100 and 200 instructions. A more thorough analysis might reveal that 100 kilobit/sec rate may be possible, but certainly difficult. A faster MPU and additional hardware might be required. Another tradeoff that can be made is memory for speed; that is, table look-up may be used in some cases to reduce the number of instructions required to be executed.

7.0 REFERENCES

1. "Use of a Microprocessor To Implement an ADCCP Protocol (Federal Standard 1003)" Delta Information Systems, Inc. July, 1980

2. Malcom Easton, "Batch Throughput Efficiency of ADCCP/ HDLC/SDLC Selective Reject Protocols" Data Communications pp 187-195, February 1980.

APPENDIX A
OPERATING SYSTEM

The design of the operating system (OS) is important because it can have a significant impact on the time required to switch the processor among concurrent processes and to handle interrupts. The approach taken makes use of the "standard" WAIT and SIGNAL primitives together with event variables. No attempt has been made to design a complete operating system; only those routines required to handle the concurrent processes are included.

Each software process is defined to be in one of three states:

ACTIVE (RUNNING) STATE - Executing computer instructions

BLOCKED STATE - Waiting for the occurrence of an event in another process

READY STATE - Waiting for processor to run

State Tranisitons are illustrated in the state diagram of Figure A-1

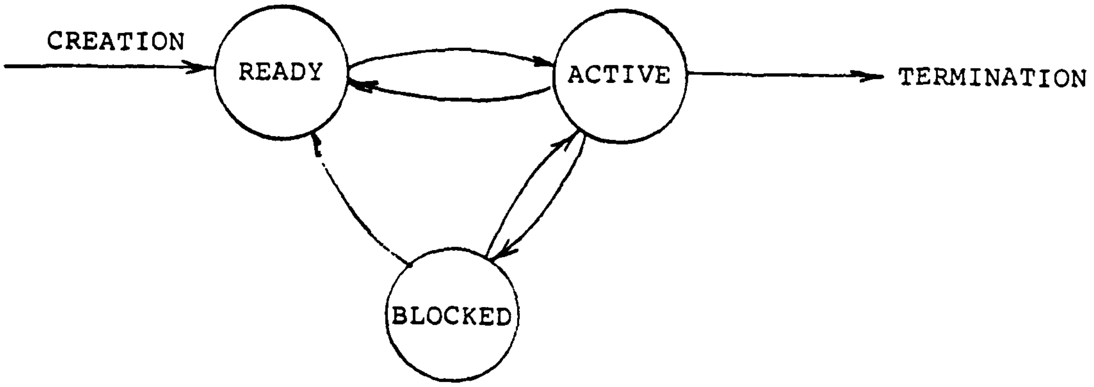


Figure A-1 Process State Diagram

Processes may be created or terminated by other processes, and each process is defined to be either running or on the blocked queue or the ready queue depending on its state. Associated with each process is a process control block (PCB) which contains an area to save the CPU's registers when the process is interrupted, a pointer to the next PCB in the queue, and the process' priority.

If the process currently running become blocked, it is changed from the ACTIVE state to the BLOCKED state via the WAIT routine. For example, if the receive process is executing instructions and wishes to obtain a byte of data from the LSI interface chip buffer, the process tests the event variable RDAFLG to determine whether or not the byte of data is available. If available, the process continues; if not, the WAIT routine is called to save the receive process registers, insert the receive process into the blocked queue, and get a new process from the ready queue, restore the registers of the new process, and run the new process. The receive process continues where it left off after an interrupt from the LSI chip signals that a byte of data is available, the interrupt service routine services the interrupts, and the receive process is moved to the running state via the SIGNAL routine. The SIGNAL routine removes the receive process from the blocked queue and restores it to the running state. Any process is blocked and restored in this way by the WAIT and SIGNAL routines respectively, and by the appropriate interrupt service routine.

The 6800 has but one interrupt input that is maskable.

This means that unless some additional hardware is used, all interrupt lines must be logically ORed and fed to the CPU via the single interrupt input, $\overline{\text{IRQ}}$. This requires that the interrupt service routine poll the various devices that might cause an interrupt to determine what device actually did cause the interrupt. The order in which the devices are polled determines the interrupt priority. The following five events cause interrupts from the 6856 protocol chip:

RDA - Received data available

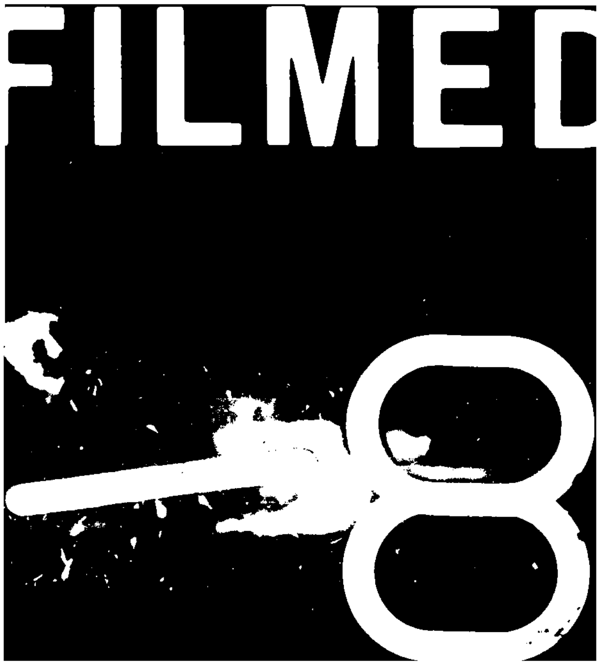
ROVR - Receiver overrun (data was not read from buffer before new byte was loaded)

TOR - Transmitter overrun

TBMT - Transmitter buffer empty

TUR - Transmitter underrun (data was not loaded in transmitter buffer in time to transmit)

In addition, two timers are assumed to be part of the design, one to provide a time slicing function to interrupt a running process periodically to give the CPU to a different process, and a time-out timer to indicate an overdue response. These two functions may be provided by one timer and appropriate software or by two separate timers.


FILMED
8